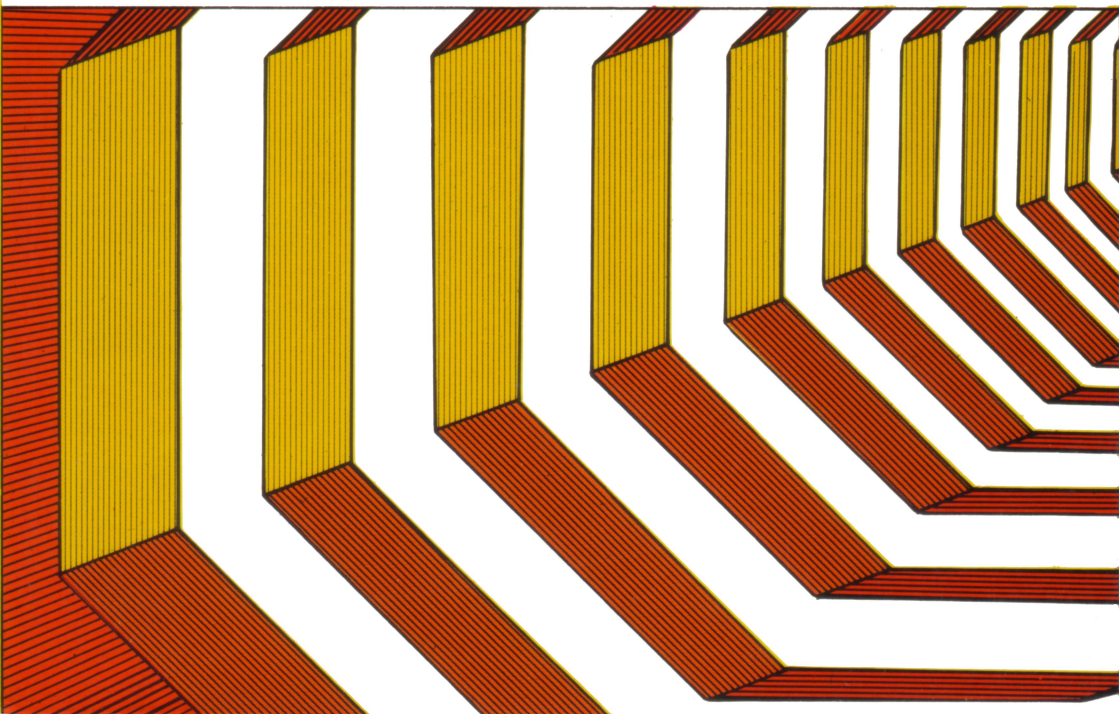
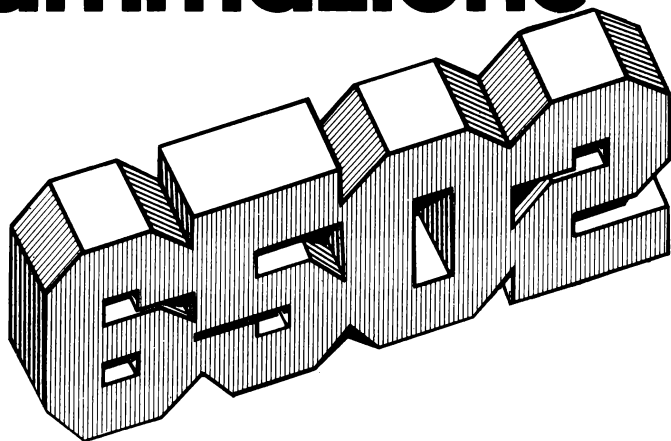
Programmazione
del 6502
EDIZIONE ITALIANA
RODNAY ZAKS
GRUPPO EDITORIALE JACKSON
JACKSON

# Programmazione del 6502

di
Rodnay
Zaks

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

# PREFAZIONE

Questo libro si propone di essere un testo autosufficiente e completo per insegnare la programmazione, impiegando il 6502.

Può essere utilizzato da chi in precedenza non ha mai programmato ed è utile anche per coloro che impiegano il 6502.

Alle persone che hanno già programmato, questo libro insegnerà tecniche di programmazione basate sulle caratteristiche specifiche del 6502. Il testo comprende le tecniche, a livello elementare ed intermedio, richieste per un effettivo inizio alla programmazione.

Il libro ha lo scopo di fornire un vero livello di competenza alle persone che desiderano programmare impiegando questo microprocessore. Naturalmente nessun libro insegnerà effettivamente come programmare, finchè non si eseguono delle applicazioni pratiche. Tuttavia questo libro guiderà il lettore al punto di poter programmare da solo e risolvere problemi semplici, od anche moderatamente complessi, impiegando un microcalcolatore.

Il libro si basa sull'esperienza acquisita dell'autore nell'insegnamento della programmazione dei microcalcolatori a più di 1000 persone.

I lettori che hanno già imparato la programmazione possono saltare il capitolo di introduzione. Per gli altri che non hanno mai programmato, la parte finale di qualche capitolo può richiedere una seconda lettura. Questo libro è stato progettato per introdurre sistematicamente il lettore a tutti i concetti base e alle tecniche richieste per la costruzione di programmi a complessità crescente. Quindi si consiglia vivamente di seguire l'ordine dei capitoli. Inoltre, per ottenere effettivi risultati, è importante che il lettore si sforzi di risolvere più esercizi possibili. La difficoltà contenuta negli esercizi è stata accuratamente graduata. Essi servono per verificare che il materiale presentato sia realmente compreso. Senza l'esecuzione di esercizi di programmazione non sarà possibile raggiungere pienamente il fine didattico che il libro si propone. Diversi esercizi possono richiedere tempo, come l'esercizio di moltiplicazione, per esempio.

Comunque, eseguendoli, si programmerà effettivamente e si *imparerà eseguendo.* Questo è indispensabile.

Per coloro che al termine di questo libro avranno acquisito "piacere" per la programmazione, è disponibile un volume ulteriore: "Applicazioni del 6502".

In questa serie, esistono altri libri per la programmazione di altri microprocessori utilizzati comunemente.

Per coloro che desiderano sviluppare le loro conoscenze hardware si suggeriscono i libri "Microprocessori" e "Tecniche di interfacciamento dei Microprocessori".

I contenuti di questo libro sono stati attentamente controllati e sono quindi affidabili. Comunque, inevitabilmente, si troveranno errori di tipo tipografico o di altro tipo.

L'autore sarà grato per qualsiasi segnalazione da parte di lettori attenti cosicchè le future edizioni possano beneficiare della loro esperienza. Qualunque altro suggerimento destinato a migliorare il libro, sarà apprezzato.

## PREFAZIONE ALLA SECONDA EDIZIONE

**Questa seconda edizione ha consentito di aumentare il libro di circa 100 pagine, con la maggior parte del nuovo materiale aggiunto ai Capitoli 1 e 9, cioè agli estremi; infatti il Capitolo 1 è quello di introduzione mentre il Capitolo 9 è dedicato alle ultime informazioni sulle strutture** dei dati.

**Nel corso del libro sono stati introdotti dei miglioramenti aggiuntivi;** in particolare l'autore desidera ringraziare i numerosi lettori dell'edizione precedente che hanno contribuito ad importanti suggerimenti migliorativi.

**Un ringraziamento particolare è rivolto a Eric Martinot e Chris Williams per i loro contributi agli esempi complessi di programmazione del Capitolo 9 ed a Daniel J. David per i numerosi miglioramenti suggeriti. Numerose variazioni e miglioramenti sono dovuti alle analisi ed ai commenti proposti da Philip K. Hooper, John Smith, Ronald Long, Charles Curley, N. Harris, John McClenon, Douglas Trusty e Fletcher Carson.**

# SOMMARIO

## CAPITOLO 5 - TECNICHE DI INDIRIZZAMENTO



## CAPITOLO 6 - TECNICHE INGRESSO/USCITA



## CAPITOLO 7 - DISPOSITIVI INGRESSO/USCITA



## CAPITOLO 8 - ESEMPI DI APPLICAZIONE

CAPITOLO 1

# CONCETTI DI BASE

## INTRODUZIONE

Questo capitolo introdurrà i concetti di base e le definizioni relative alla programmazione di calcolatori. Il lettore già familiare con questi concetti può scorrer velocemente i contenuti di questo capitolo e poi passare al secondo capitolo. Comunque è consigliabile che anche i lettori esperti osservino i contenuti di questo capitolo di introduzione: qui sono compresi, molti concetti significativi comprendendo, per esempio il complemento a due, BCD, ed altre rappresentazioni.

## COS'E' LA PROGRAMMAZIONE?

Dato un problema occorre innanzi tutto escogitare un metodo di soluzione. Questa soluzione è espressa come una procedura di fasi successive chiamata *algoritmo.* Un algoritmo è quindi una specificazione fase-per-fase della soluzione da dare ad un problema. È necessario inoltre terminare la soluzione di un numero finito di fasi. Questo algoritmo può essere espresso in qualsiasi linguaggio.

Un algoritmo tipico può essere per esempio:

1 - inserire la chiave nella toppa
2 - girare la chiave a sinistra per un giro completo
3 - impugnare la maniglia
4 - girare la maniglia a sinistra e spingere la porta

A questo punto, se questo algoritmo è corretto per il tipo di serratura considerato, la porta si aprirà. Questa procedura formata da 5 fasi, viene considerata come un algoritmo per l'apertura della porta.

Una volta che la soluzione del problema è stata espressa sotto forma di un algoritmo occorre che questo algoritmo venga eseguito dal calcolatore. Sfortunatamente è ora ben noto il fatto che i calcolatori non possono capire, ed eseguire, il linguaggio comunemente impiegato. Questo a causa dell'*ambiguità sintattica* di tutti i comuni linguaggi umani. Solo una parte ben definita del linguaggio naturale può essere "capita" dal calcolatore. Questa parte è chiamata *linguaggio di programmazione.*

La conversione di un algoritmo in una sequenza di istruzioni in linguaggio di programmazione è detta programmazione. Per essere più specifici, l'attuale fase di traduzione dell'algoritmo in linguaggio di programmazione, è chiamato *codifica.*

In realtà la programmazione fa riferimento non solo alla codifica ma comprende il progetto globale dei programmi e le "strutture dati" che realizzano l'algoritmo.

La programmazione effettiva richiede non solo la comprensione delle tecniche di realizzazione possibili per gli algoritmi convenzionali ma anche l'abilità di impiego di tutte le caratteristiche hardware del calcolatore come i registri interni, la memoria ed i dispositivi periferici, più un impiego costruttivo di opportune strutture di dati. Queste tecniche saranno sviluppate nei capitoli successivi.

La programmazione richiede anche una severa disciplina di documentazione, in modo che i programmi siano comprensibili da altre persone oltre all'autore. La documentazione deve essere interna ed esterna al programma.

La documentazione interna del programma fa riferimento ai commenti introdotti nel corpo di un programma, allo scopo di spiegare il suo modo di operare.

La documentazione esterna consiste nei documenti di progetto che sono separati dal programma: spiegazioni scritte, manuali e diagrammi di flusso.

## DIAGRAMMI DI FLUSSO

Esiste quasi sempre una fase intermedia fra *l'algoritmo* ed il *programma.* Questa fase utilizza i *diagrammi di flusso.* Un diagramma di flusso è semplicemente una rappresentazione simbolica dell'algoritmo, espressa come sequenza di blocchi contenenti le fasi dell'algoritmo. Questi blocchi sono dei rettangoli se utilizzati per *comandi* ovvero "statement eseguibili". Invece per *test* come: se l'informazione X è vera esegue l'azione A, altrimenti B si utilizzano dei blocchi a forma rombica. Anzichè presentare qui una definizione formale dei diagrammi di flusso, questi saranno introdotti e discussi in seguito quando si considereranno i programmi.

Il metodo dei diagrammi di flusso è una fase intermedia altamente raccomandabile tra la specificazione dell'algoritmo e la codifica effettiva della soluzione. Si fa notare che forse solo il 10% dei programmatori può scrivere con successo un programma senza l'utilizzo del diagramma di flusso. Sfortunatamente è stato anche osservato che il 90% di questa popolazione crede di appartenere a questo 10%!

Il risultato è questo: mediamente l'80% dei programmi si interrompono la prima volta che vengono eseguiti su un calcolatore. (Naturalmente questi numeri non si propongono di essere accurati). In breve, la maggior parte dei non iniziati alla programmazione raramente intravede la necessità di disegnare un diagramma di flusso. Questo si risolve normalmente in programmi "non puliti" ovvero errati. Essi devono così impiegare una grande quantità di tempo per provare e correggere i loro programmi (questa fase è detta di *collaudo* o *debugging*).

Si raccomanda vivamente quindi di passare attraverso la disciplina dei diagrammi di flusso in tutti i casi. Questo richiederà una piccola quantità di tempo addizionale prima della codifica ma normalmente si risolverà in un programma chiaro che verrà eseguito correttamente e velocemente. Una volta che è stata ben compresa la tecnica dei diagrammi di flusso una piccola percentuale di programmatori sarà in grado di eseguire mentalmente questa fase senza trascriverla su carta. Sfortunatamente in tali casi i programmi che essi scriveranno saranno normalmente difficili da capire a chiunque senza la documentazione fornita dai diagrammi di flusso. Come risultato si raccomanda universalmente di impiegare la disciplina dei diagrammi di flusso come una stringente disciplina di programmazione per qualsiasi programma significativo. Molti esempi saranno forniti in seguito nel corso del libro.

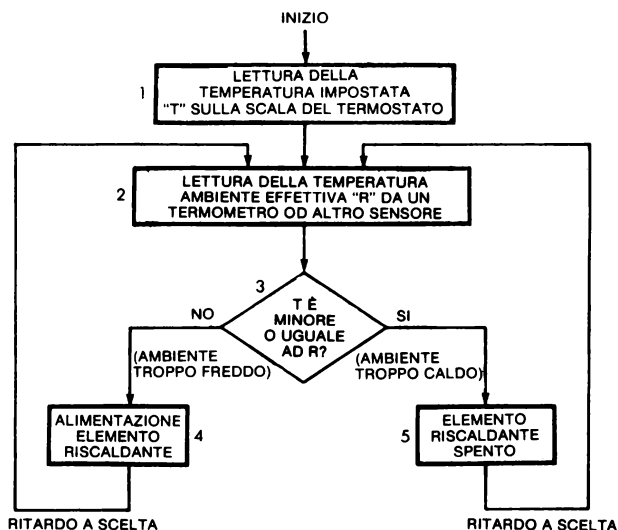



**Figura 1-1: Un diagramma di flusso di un sistema di termostatazione**

## RAPPRESENTAZIONE DELL'INFORMAZIONE

Tutti i calcolatori manipolano informazione, sotto forma di numeri o di caratteri. Si esamineranno di seguito le rappresentazioni esterna ed interna dell'informazione di un calcolatore.

### *RAPPRESENTAZIONE INTERNA*

Tutte le informazioni sono immagazzinate in un calcolatore come gruppi di bit. Un *bit* significa un *digit binario* cioè "0" oppure "1". A causa delle limitazioni dell'elettronica convenzionale, la sola rappresentazione pratica dell'informazione impiega la logica a due stati, cioè la rappresentazione degli stati "0" ed "1". Ne risulta che virtualmente tutta l'elaborazione dell'informazione attualmente è eseguita in formato binario. Nel caso dei microprocessori in generale, e del 6502 in particolare, questi bit sono strutturati in gruppi di 8. Un gruppo di 8 bit è chiamato un *byte.*

Un gruppo di quattro bit è chiamato un *nibble.*

Si esamini ora come l'informazione è rappresentata internamente nel suo formato binario. Due entità devono essere rappresentate all'interno del calcolatore. La prima è il programma, che è una sequenza di istruzioni. La seconda sono i dati sui quali esso opera, e che possono comprendere numeri di un testo alfanumerico. Si esaminino queste tre rappresentazioni.

#### Rappresentazione del programma

Tutte le istruzioni sono rappresentate internamente come byte singoli o multipli. Una cosiddetta "istruzione breve" è rappresentata da un singolo byte. Un'istruzione più lunga sarà rappresentata da due o più byte. Poichè il 6502 è un microprocessore ad 8 bit, esso preleva i byte sequenzialmente dalla sua memoria. Perciò un'istruzione a singolo byte è sempre potenzialmente di esecuzione più veloce di un'istruzione a due o tre byte. Si vedrà in seguito che questa è un'importante caratteristica del set di istruzioni di qualsiasi microprocessore ed in particolare del 6502 dove è stato fatto uno sforzo speciale per fornire più istruzioni a singolo byte possibile in modo da migliorare l'efficienza di esecuzione del programma. Comunque la limitazione ad 8 bit in lunghezza, si risolve in importanti limitazioni che verranno sottolineate. Questo è un esempio classico del compromesso tra efficienza di velocità e flessibilità nella programmazione.

La rappresentazione binaria delle istruzioni è dettata dal costruttore ed il 6502, come qualsiasi altro microprocessore, viene equipaggiato con

un set di istruzioni fisso. Queste istruzioni sono definite dal costruttore e sono elencate alla fine di questo libro. Qualsiasi programma sarà espresso come sequenza di queste istruzioni binarie. L'effettiva codifica binaria delle istruzioni del 6502 è rappresentata nel Capitolo 4.

## Rappresentazione di dati numerici

La rappresentazione di numeri non è sufficientemente immediata ed, in alcuni casi, deve essere distinta. Si devono innanzi tutto rappresentare i numeri interi. Si devono anche rappresentare numeri con segno, cioè positivi o negativi, ed infine si deve essere in grado di rappresentare i numeri decimali. Si considereranno ora queste richieste e le soluzioni possibili.

La rappresentazione di numeri interi deve essere eseguita impiegando una rappresentazione *binaria diretta.* La rappresentazione binaria diretta è semplicemente la rappresentazione del valore decimale di un numero nel sistema binario. Nel sistema binario il bit più a destra rappresenta 2 elevato alla potenza 0. Quello successivo a sinistra rappresenta 2 alla potenza 1, il successivo rappresenta 2 alla potenza 2 ed il bit più a sinistra rappresenta 2 alla potenza 7 = 128:

$$b_7b_6b_5b_4b_3b_2b_1b_0$$

rappresenta

$$b_72^7 + b_62^6 + b_52^5 + b_42^4 + b_32^3 + b_22^2 + b_12^1 + b_02^0$$

Le potenze di 2 sono:

$$2^7 = 128,\ 2^6 = 64,\ 2^5 = 32,\ 2^4 = 16,\ 2^3 = 8,\ 2^2 = 4,\ 2^1 = 2,\ 2^0 = 1$$

La rappresentazione binaria dei numeri è analoga a quella decimale, nella quale "123" rappresenta:

$$\begin{array}{r} 1\times100=100 \\ +2\times\ \ 10=\ \ 20 \\ +3\times\ \ \ 1=\ \ \ 3 \\ \hline =123 \end{array}$$

Si noti che $100 = 10^2$, $10 = 10^1$, $1 = 10^0$.
In questa "notazione posizionale" ogni cifra rappresenta una potenza di 10. Nel sistema binario ogni digit o "bit" rappresenta una potenza di 2, invece di una potenza di 10 del sistema decimale.

Esempio: "00001001"; in binario rappresenta:

$$\begin{array}{rl} 1\times\ \ 1=1 & (2^0) \\ 0\times\ \ 2=0 & (2^1) \\ 0\times\ \ 4=0 & (2^2) \\ 1\times\ \ 8=8 & (2^3) \\ 0\times\ 16=0 & (2^4) \\ 0\times\ 32=0 & (2^5) \\ 0\times\ 64=0 & (2^6) \\ 0\times 128=0 & (2^7) \\ \hline \text{in decimale}\quad =9 & \end{array}$$

Si considerino alcuni esempi:

"10000001" rappresenta:

$$\begin{array}{r} 1\times\ \ 1=\ \ 1 \\ 0\times\ \ 2=\ \ 0 \\ 0\times\ \ 4=\ \ 0 \\ 0\times\ \ 8=\ \ 0 \\ 0\times\ 16=\ \ 0 \\ 0\times\ 32=\ \ 0 \\ 0\times\ 64=\ \ 0 \\ 1\times 128=128 \\ \hline \text{in decimale}\quad =129 \end{array}$$

"10000001" rappresenta perciò il numero decimale 129.

Esaminando la rappresentazione binaria dei numeri, si comprenderà perchè i bit sono numerati da 0 a 7, andando da destra a sinistra. Il bit 0 è "$b_0$" e corrisponde a $2^0$. Il bit 1 è "$b_1$" e corrisponde a $2^1$ ecc.

La Fig. 1-2 mostra gli equivalenti binari dei numeri da 0 a 255.

**Esercizio 1.1:** *Qual'è il valore decimale di "111111100"?*

*Da decimale a binario*

Inversamente, si calcoli l'equivalente binario del decimale "11":

$$\begin{array}{rll} 11\div2=5 \text{ resto } 1 & \rightarrow 1 & \text{(LSB)} \\ 5\div2=2 \text{ resto } 1 & \rightarrow 1 & \\ 2\div2=1 \text{ resto } 0 & \rightarrow 0 & \\ 1\div2=0 \text{ resto } 1 & \rightarrow 1 & \text{(MSB)} \end{array}$$

| Decimale | Binario | Decimale | Binario |
|---|---|---|---|
| 0 | 00000000 | 32 | 00100000 |
| 1 | 00000001 | 33 | 00100001 |
| 2 | 00000010 | • | |
| 3 | 00000011 | • | |
| 4 | 00000100 | • | |
| 5 | 00000101 | 63 | 00111111 |
| 6 | 00000110 | 64 | 10000001 |
| 7 | 00000111 | 65 | 01000001 |
| 8 | 00001000 | • | |
| 9 | 00001000 | • | |
| 10 | 00001010 | 127 | 01111111 |
| 11 | 00001011 | 128 | 10000000 |
| 12 | 00001100 | 129 | 10000001 |
| 13 | 00001101 | | |
| 14 | 00001110 | • | |
| 15 | 00001111 | • | |
| 16 | 00010000 | • | |
| 17 | 00010001 | • | |
| • | | | |
| • | | | |
| • | | 254 | 11111110 |
| 31 | 00011111 | 255 | 11111111 |

Figura 1-2: Tabella decimale binario

Il binario equivalente è quindi 1011 (lettura corretta della colonna dal basso all'alto).
L'equivalente binario di un numero decimale può essere ottenuto mediante divisioni successive per 2 finchè non si ottiene un quoziente 0.

**Esercizio 1.2:** *Qual'è l'equivalente binario di 257?*

**Esercizio 1.3:** *Si converta 19 in binario e quindi nuovamente in decimale.*

*Operazioni sui dati binari*

Le regole aritmetiche sui numeri binari sono immediate.

Le regole per l'addizione sono:

$$\begin{aligned} 0+0 &= 0 \\ 0+1 &= 1 \\ 1+0 &= 0 \\ 1+1 &= (1)0 \end{aligned}$$

dove (1) indica un "riporto" (carry) di 1 (si noti che "10" è l'equivalente binario del decimale "2"). La sottrazione binaria verrà eseguita "sommando il complemento". Esempio:

```
   (2)      10
 + (1)    + 01
 -----    ----
 = (3)      11
```

L'addizione viene eseguita in modo perfettamente uguale al caso decimale, sommando colonna per colonna, da destra a sinistra:

Sommando la colonna più a destra:

```
       10
     + 01
 ---------
(0 + 1 =  1. Nessun riporto.)
```

Sommando la colonna successiva:

```
   10
 + 01
 ----
   11    (1 + 0 = 1. Nessun riporto.)
```

**Esercizio 1.4:** *Si calcoli 5 + 10 in binario e si verifichi che il risultato è 15.*

Altrimenti esempi di addizione binaria:

```
  0010     (2)          0011     (3)
 +0001     (1)         +0001     (1)
 -------------         -------------
 =0011     (3)         =0100     (4)
```

L'ultimo esempio spiega il ruolo del riporto.

Si osservino i bit di estrema destra: 1 + 1 = (1) 0
Si genera un riporto di 1 che deve essere sommato ai bit successivi:

```
    001 — la colonna 0 è stata appena sommata
  + 000 —
  +   1   (riporto)
 -------
 = (1)0 — dove (1) indica un nuovo riporto
          nella colonna 2.
```

Il risultato finale è: 0100

Un altro esempio:

```
   0111      (7)
  +0011     +(3)
 ----------------
   1010   = (10)
```

In questo esempio si genera un riporto fino alla colonna di estrema sinistra.

**Esercizio 1.5:** *Si calcoli il risultato di:*

```
  1111
 +0001
 -----
 =?
```

*Il risultato può essere contenuto in 4 bit?*

Con 8 bit è perciò possibile rappresentare direttamente i numeri da 00000000 a 11111111, cioè da 0 a 255. Si possono osservare immediatamente due ostacoli. Primo si stanno rappresentando solo numeri positivi. Secondo, la grandezza di questi numeri è limitata a 255 se si impiegano solo 8 bit.
Nell'ordine si considerano entrambi questi problemi.

*Binario con Segno*

In una rappresentazione binaria con segno il bit più a sinistra è impiegato per indicare il segno del numero. Tradizionalmente "0" è impiegato per denotare un numero *positivo* mentre "1" è impiegato per denotare un numero *negativo.*
Ora "11111111" rappresenterà —127, mentre "01111111" rappresenterà + 127. Si possono ora rappresentare numeri positivi e negativi ma si è ridotta la grandezza massima di questi numeri a 127.

Esempio: "0000 0001" rappresenta + 1 (lo "0" di testa è il "+", seguito da "0000 0001" = 1).

"1000 0001" è — 1 (l'"1" di testa è il "—").

**Esercizio 1.6:** *Qual'è la rappresentazione di "- 5" in binario con segno?*

Si consideri ora il problema della grandezza: allo scopo di rappresentare numeri più grandi sarà necessario utilizzare un maggior numero di

bit. Per esempio se si utilizzano 16 bit (due byte) per la rappresentazione di numeri si è in grado di rappresentare numeri da — 32 k a + 32 k in binario con segno (1 k nel gergo del calcolatore rappresenta 1024). Se questa grandezza è ancora troppo piccola si utilizzeranno 3 o più byte. Se si desidera rappresentare interi molto grandi, sarà necessario utilizzare un numero maggiore di byte interni per la loro rappresentazione. Questa è la ragione per cui il più semplice BASIC, od altri linguaggi, forniscono solo una precisione limitata per gli interi.

Si consideri ora un altro problema, quello dell'efficienza di velocità. Si consideri l'esecuzione di un'addizione nella rappresentazione binaria con segno precedentemente introdotto. Si sommi "—5" e "+ 7".

+ 7 è rappresentato da 00000111
— 5 è rappresentato da 10000101
La somma binaria è 10001100, ovvero — 12

Questo risultato non è esatto. Il risultato corretto sarebbe + 2. In altre parole l'addizione binaria di numeri binari con segno non opera in modo corretto. Questo è fastidioso. Chiaramente il calcolatore non deve soltanto rappresentare l'informazione ma deve anche eseguire operazioni aritmetiche su questa.

La soluzione a questo problema è chiamata rappresentazione *complemento a due,* che sarà impiegata invece della rappresentazione *binaria con segno.* Allo scopo di introdurre il complemento a due si introdurrà una fase intermedia: *il complemento ad uno.*

*Complemento ad uno*

Nella rappresentazione in complemento ad uno tutti gli interi sono rappresentati nel loro formato binario corretto. Per esempio "+ 3" è rappresentato come comunemente da 00000011. Comunque il suo complemento "— 3" è ottenuto mediante complementazione di ogni bit della rappresentazione originaria. Ogni 0 è trasformato in un 1 ed ogni 1 è trasformato in uno 0. Nell'esempio considerato la rappresentazione in complemento ad uno di "— 3" sarà 11111100.

Un'altro esempio:

+ 2 è 00000010
—2 è 11111101

Si noti che i numeri positivi iniziano con 0 e quelli negativi con 1

**Esercizio 1.7:** *La rappresentazione di + 6 è 00000110. Qual'è la rappresentazione di meno 6 in complemento ad uno?*

Si esegua ora la prova convenzionale, cioè si sommi meno 4 e più 6:

```
              −4 è 11111011
              + 6 è 00000110
              ──────────────
la somma è    (1)   00000001  ovvero "1"
                              più un riporto.
```

Il "risultato corretto" sarebbe "2", ovvero "00000010".

Ripproviamo:

```
              −3 è 11111100
              −2 è 11111101
              ──────────────
la somma è    (1)   00000001
```

o "1", più un riporto. Il risultato corretto dovrebbe essere "— 5". La rappresentazione di "— 5" è 11111010. Non ha funzionato.

Inoltre questa rappresentazione rappresenta numeri positivi e negativi. Comunque il risultato di un'addizione ordinaria non esce correttamente. Si considererà quindi un'ulteriore rappresentazione. Questa è un'evoluzione dal complemento ad uno ed è chiamata rappresentazione in complemento a due.

*Rappresentazione in complemento a due*

Nella rappresentazione in complemento a due i numeri positivi sono rappresentati come al solito, in binario con segno, proprio come in complemento ad uno. La differenza risiede nella rappresentazione di *numeri negativi.* Un numero negativo, rappresentato in complemento a due, è ottenuto dal primo *sommando uno* al complemento ad uno. Si consideri un esempio: + 3 è rappresentato in forma binaria con segno da 00000011. La sua rappresentazione in complemento ad uno è 11111100. Il complemento a due è ottenuto aggiungendo uno. Esso è 11111101. Se non si considera il riporto il risultato è 00000001, cioè 1 in decimale. Questo è il risultato corretto.

Si verifichi un'addizione:

```
     (3)        00000011
   + (5)      + 00000101
   ─────────────────────
   = (8)      = 00001000
```

Il risultato è corretto.

Si verifichi una sottrazione:

```
     (3)      00000011
    (−5)    + 11111011
  --------------------
            = 11111110
```

Si determina il risultato calcolando il complemento a due:

```
il complemento ad uno di 11111110 è       00000001
                      Aggiungendo 1     +        1
                                       ----------
perciò il complemento a 2 è               00000010   cioè + 2
```

Il risultato precedente, "11111110" rappresenta — 2. È corretto.

Si è quindi trovato che i risultati di addizioni e sottrazioni sono corretti (non considerando il riporto). Sembra che il complemento a due operi correttamente!

**Esercizio 1.8:** *Qual'è la rappresentazione in complemento a due di "+ 127"?*

**Esercizio 1.9:** *Qual'è la rappresentazione in complemento a due di "- 128"?*

Si consideri ora la somma di + 4 e — 3 (la sottrazione si esegue sommando il complemento a due):

```
                    + 4 è 00000100
                    − 3 è 11111101
                   ---------------
Il risultato è     (1)    00000001
```

dove (1) indica il riporto. Senza fornire una dimostrazione matematica completa si è stabilito che questa rappresentazione opera correttamente. In complemento a due è possibile sommare e sottrare numeri con segno. Impiegando le regole usuali dell'addizione binaria si ottiene un risultato corretto, compreso il segno. Il riporto viene trascurato. Questo è un vantaggio molto importante, infatti, in caso contrario, si dovrebbe correggere il segno ogni volta, con un tempo di esecuzione notevolmente superiore.

Per completezza si può affermare che la rappresentazione in complemento a due è la più conveniente per i processori più semplici quali i microprocessori. Sui processori più complessi si possono utilizzare altre rappresentazioni. Per esempio si può utilizzare la rappresentazione in complemento ad uno, ma essa richiede una circuiteria speciale per "correggere il risultato".

D'ora in poi tutti gli interi con segno verranno, in modo implicito, rappresentati internamente con la notazione in complemento a due. La Fig. 1-3 rappresenta una tabella dei numeri in complemento a due.

**Esercizio 1.10:** *Quali sono i numeri più piccolo e più grande che si possono rappresentare con la notazione in complemento a due impiegando solo un byte?*

**Esercizio 1.11:** *Si calcoli il complemento a due di 20. Quindi si calcoli il complemento a due del risultato. Si ottiene ancora 20?*

Di seguito vengono riportati degli esempi per dimostrare le regole della notazione in complemento a due. In particolare C rappresenta un possibile riporto (o prestito). (Esso è il bit 8 del risultato).

Invece V rappresenta un overflow del complemento a due, cioè indica una variazione "accidentale" del segno del risultato a causa di numeri troppo grandi. Si tratta sostanzialmente di un riporto interno dal bit 6 al bit 7 (il bit del segno).

Questo verrà chiarito in seguito.

Si considera ora il ruolo del riporto "C" e dell'overflow "V".

*Il riporto C*

Ecco un esempio di un riporto:

```
  (128)         10000000
+ (129)       + 10000001
------------------------
  (257)  = (1)  00000001
```

dove (1) indica un riporto.

Il risultato richiede un nono bit (bit "8" poichè quello di estrema destra è il bit "0"). Questo è il bit carry.

Si assume quindi che il carry sia il nono bit del risultato e si riconosce che il risultato è 100000001 = 257.

Comunque il carry deve essere riconosciuto e manipolato accuratamente. All'interno del microprocessore, i registri utilizzati per conservare l'informazione sono larghi generalmente solo otto bit. Nella memorizzazione del risultato saranno conservati soltanto i bit da 0 a 7.

Quindi un riporto richiede sempre un'azione speciale: esso deve essere rivelato da istruzioni particolari e quindi deve essere elaborato. L'elaborazione del riporto può significare, a seconda dei casi, la sua memorizzazione (con un'istruzione particolare), non tenerne conto oppure decidere

| + | Codice in complemento a 2 | — | Codice in complemento a 2 |
|---|---|---|---|
| +127 | 01111111 | —128 | 10000000 |
| +126 | 01111110 | —127 | 10000001 |
| +125 | 01111101 | —126 | 10000010 |
| . . . | | —125 | 10000011 |
| | | . . . | |
| + 65 | 01000001 | — 65 | 10111111 |
| + 64 | 01000000 | — 64 | 11000000 |
| + 63 | 00111111 | — 63 | 11000001 |
| . . . | | . . . | |
| + 33 | 00100001 | — 33 | 11011111 |
| + 32 | 00100000 | — 32 | 11100000 |
| + 31 | 00011111 | — 31 | 11100001 |
| . . . | | . . . | |
| + 17 | 00010001 | — 17 | 11101111 |
| + 16 | 00010000 | — 16 | 11110000 |
| + 15 | 00001111 | — 15 | 11110001 |
| + 14 | 00001110 | — 14 | 11110010 |
| + 13 | 00001101 | — 13 | 11110011 |
| + 12 | 00001100 | — 12 | 11110100 |
| + 11 | 00001011 | — 11 | 11110101 |
| + 10 | 00001010 | — 10 | 11110110 |
| + 9 | 00001001 | — 9 | 11110111 |
| + 8 | 00001000 | — 8 | 11111000 |
| + 7 | 00000111 | — 7 | 11111001 |
| + 6 | 00000110 | — 6 | 11111010 |
| + 5 | 00000101 | — 5 | 11111011 |
| + 4 | 00000100 | — 4 | 11111100 |
| + 3 | 00000011 | — 3 | 11111101 |
| + 2 | 00000010 | — 2 | 11111110 |
| + 1 | 00000001 | — 1 | 11111111 |
| + 0 | 00000000 | | |

Figura 1-3: Tabella dei complementi a due

che si tratta di un errore (se il risultato più grande consentito è "11111111").

*Overflow V*

Consideriamo un esempio di overflow:

In questo caso è stato generato un riporto interno dal bit 6 al bit 7. Questo è ciò che si chiama un overflow.

Accidentalmente il risultato è negativo. Occorre rivelare situazioni di questo tipo in modo da intervenire.

Si consideri un'altra situazione:

```
            11111111   (– 1)
          + 11111111 + (– 1)
      ------------------------
  = (1)     10000001 = (– 2)
     ▼
   carry
```

Anche in questo caso è stato generato un riporto interno dal bit 6 al bit 7, ma questo ha generato a sua volta un riporto dal bit 7 al bit 8 (il "Carry" C formale, considerato al paragrafo precedente). Le regole dell'aritmetica in complemento a 2 specificano che questo riporto dovrebbe essere ignorato. Il risultato è quindi corretto.

Questa è una conseguenza del fatto che il riporto dal bit 6 al bit 7 non cambia il segno.

Questa non è una condizione di *overflow*. Quando si opera su numeri negativi l'overflow non è semplicemente un riporto dal bit 6 al bit 7. Si consideri un ulteriore esempio:

```
            11000000   (– 64)
          + 10111111   (– 65)
      ------------------------
  = (1)     01111111   (+ 127)
     ▼
   carry
```

Questa volta non si è verificato un riporto interno dal bit 6 al bit 7, ma si è verificato un riporto esterno. Il risultato non è corretto in quanto è stato cambiato il bit 7. In questo caso è quindi indispensabile indicare una condizione di overflow.

L'overflow si verificherà nelle quattro situazioni:

1 - somma di numeri positivi troppo grandi

2 - somma di numeri negativi troppo grandi in valore assoluto

3 - sottrazione di un numero positivo molto grande da un numero negativo molto grande in valore assoluto

4 - sottrazione di un numero negativo molto grande in valore assoluto da un numero positivo molto grande.

Precisiamo la definizione di overflow precedentemente fornita.

Tecnicamente l'indicatore di overflow, uno speciale bit riservato per questo scopo, cioè un "flag", sarà posto ad uno quando si verifica un riporto dal bit 6 al bit 7 ma non un riporto esterno, oppure quando non c'è riporto dal bit 6 al bit 7 ma si verifica un riporto esterno. Questo indica che il bit 7, cioè il segno del risultato, è stato cambiato accidentalmente. Come si ricorderà, il flag di overflow è posto ad 1 dall'OR ESCLUSIVO del bit 7 entrante ed uscente (il bit segno). Praticamente ogni microprocessore è dotato di un flag di overflow speciale per rivelare automaticamente questa condizione, che richiede un'azione correttiva.

Overflow indica che il risultato di un'addizione o di una sottrazione richiede più bit di quelli disponibili nel registro standard di otto bit, utilizzato per contenere il risultato.

*Il Carry e l'Overflow*

I bit carry ed overflow vengono denominati "flag". Essi sono disponibili su tutti i microprocessori e, nel corso del capitolo successivo, si apprenderà come utilizzarli per la programmazione effettiva. Questi due indicatori sono posizionati in un registro speciale denominato registro di "stato" o dei flag. Questo registro contiene altri indicatori le cui funzioni saranno chiarite al Capitolo 4.

*Esempi*

Si considerano ora degli esempi pratici per mostrare la funzione del carry e dell'overflow. Il simbolo V indica overflow e C carry.

Se non si verifica overflow è V = 0. Se si è verificato un overflow è V = 1 (analogamente per il carry C). Si ricordi che le regole del complemento a due specificano che il carry deve essere ignorato. (In questa sede non viene fornita la dimostrazione matematica).

**Positivo-Positivo**

| | | | |
|---|---|---|---|
| | 00000110 (+ 6) | | |
| + | 00001000 (+ 8) | | |
| = | 00001110 (+ 14) | V:0 | C:0 |

(CORRETTO)

**Positivo-Positivo con Overflow**

| | | | |
|---|---|---|---|
| | 01111111 (+ 127) | | |
| + | 00000001 (+ 1) | | |
| = | 10000000 (— 128) | V:1 | C:0 |

Il risultato non è valido perchè si è verificato in overflow

(ERRORE)

**Positivo-Negativo (risultato positivo)**

| | | | |
|---|---|---|---|
| | 00000100 (+ 4) | | |
| + | 11111110 (— 2) | | |
| = | (1)00000010 (+ 2) | V:0 | C:1 (ignorato) |

(CORRETTO)

**Positivo-Negativo (risultato negativo)**

| | | | |
|---|---|---|---|
| | 00000010 (+ 2) | | |
| + | 11111100 (– 4) | | |
| = | 11111110 (— 2) | V:0 | C:0 |

(CORRETTO)

**Negativo-Negativo**

| | | | |
|---|---|---|---|
| | 11111110 (– 2) | | |
| + | 11111010 (– 4) | | |
| = | (1)11111010 (— 6) | V:0 | C:1 (ignorato) |

(CORRETTO)

**Negativo-Negativo con overflow**

```
    10000001 (+ 127)
+   11000010  (– 62)
=(1)01000011     (67)           V:1          C:1
```

(ERRORE)

Questa volta si è verificato un "underflow", sommando due numeri negativi molto grandi in valore assoluto. Il risultato dovrebbe essere —189, che è troppo grande per essere contenuto in otto bit.

**Esercizio 1.12:** *Si completino le seguenti addizioni. Si indichi il risultato, il carry C, l'overflow V e si specifichi se il risultato è corretto o no.*

```
   10111111   (__)
  +11000001   (__)
  _________
= _________   V: __   C: ____
□ CORRETTO      □ERRORE
```

```
   00010000   (__)
  +01000000   (__)
  _________
= _________   V: __   C: ____
□ CORRETTO      □ERRORE
```

```
   11111010   (__)
  +11111001   (__)
  _________
= _________   V: __   C: ____
□ CORRETTO      □ERRORE
```

```
   01111110   (__)
  +00101010   (__)
  _________
= _________   V: __   C: ____
□ CORRETTO      □ERRORE
```

**Esercizio 1.13:** *Potete fornire un esempio di overflow sommando un numero positivo ed uno negativo? Perche?*

*Rappresentazione in Formato Fisso*

A questo punto si conosce la rappresentazione degli interi con segno. Comunque non è ancora stato risolto il problema della grandezza. Se si vogliono rappresentare numeri interi più grandi, è necessario utilizzare più byte. Per eseguire efficientemente operazioni aritmetiche, è necessario utilizzare un numero fisso di byte piuttosto che un numero variabile. Perciò, una volta scelto il numero di byte, è fissa la grandezza massima del numero che può essere rappresentato.

**Esercizio 1.14:** *Quali sono i numeri più grande e più piccolo che possono essere rappresentati in due byte impiegando la notazione in complemento a due?*

*Il problema della grandezza*

La somma di numeri è limitata dal fatto che il microprocessore opera internamente su otto bit alla volta. Questa restrizione consente di utilizzare numeri nella gamma da — 128 a + 127.
Chiaramente questo non è sufficiente per numerose applicazioni.

Per aumentare il numero di digit che possono essere rappresentati occorre utilizzare la precisione multipla. Si può utilizzre un formato a due, tre, oppure N byte.
Per esempio si consideri un formato in doppia precisione a 16 bit:

```
00000000   00000000   è "0"
00000000   00000001   è "1"

01111111   11111111   è "32767"
11111111   11111111   è "— 1"
11111111   11111110   è "— 2"
```

**Esercizio 1.15:** *Qual'è l'intero negativo più grande, in senso assoluto, che può essere rappresentato, in un formato in tripla precisione, in complemento a due?*

Comunque, questo metodo presenta degli svantaggi. Per esempio sommando due numeri occorrerà sommarli otto bit alla volta. Questo verrà spiegato al Capitolo 4 (Tecniche di Programmazione di Base). Ne risulta un'elaborazione più lenta. Inoltre questa rappresentazione impiega 16 bit per un numero qualsiasi, anche se esso potrebbe essere rappresentato con soli otto bit. Quindi è comune utilizzare 16 od anche 32 bit ma talvolta è sovrabbondante.

Si consideri ora il seguente punto importante: qualunque sia il numero N di bit scelti per la rappresentazione in complemento a due, esso è fisso. Se qualsiasi risultato o calcolo intermedio dovesse generare un numero che richiede più di N bit, alcuni bit andranno persi. Normalmente il programma conserva gli N bit di sinistra (quelli più significativi) e perde quelli di basso ordine. Questo è il troncamento del risultato.

Ecco un esempio nel sistema decimale, utilizzando una rappresentazione a sei digit:

```
   123456
×     1,2
---------
   246912
  123456
---------
= 148147,2
```

Il risultato richiede 7 digit! Il "2" dopo la virgola andrà perso ed il risultato finale sarà 148147. Si ha così un troncamento. Normalmente, non perdendo la posizione della virgola questo metodo viene utilizzato per estendere la gamma di operazioni che possono essere eseguite, a scapito della precisione.

Il problema è analogo per la rappresentazione binaria. I dettagli della moltiplicazione binaria saranno mostrati al Capitolo 4.

Questa rappresentazione in formato fisso può causare una perdita di precisione, ma questa è generalmente sufficiente per i calcoli comuni e le operazioni matematiche.

Sfortunatamente, nel caso dei calcoli contabili, non è tollerata nessuna perdita di precisione. Per esempio un venditore potrebbe non tollerare un arrotondamento del risultato di cassa. Quindi quando è essenziale la precisione del risultato è indispensabile utilizzare un'altra rappresentazione.

Normalmente la soluzione è la rappresentazione *BCD*, cioè decimale codificato binario.

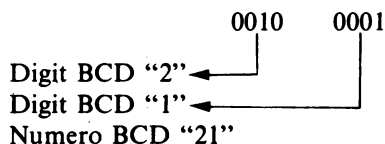
### *Rappresentazione BCD*

Il principio utilizzato nella rappresentazione di numeri in BCD è di codificare separatamente ogni digit decimale e di utilizzare tutti i bit necessari per rappresentare esattamente il numero completo. Per codificare tutti i digit da 0 a 9 sono necessari quattro bit. Tre bit consentirebbero soltanto otto combinazioni e quindi non sono sufficienti per le dieci cifre decimali. Quattro bit consentono sedici combinazioni e sono quindi

sufficienti per codificare i digit da 0 a 9. Si può anche notare che nella rappresentazione BCD saranno inutilizzati sei dei codici possibili (Vedere Fig. 1.4).
Questo rappresenterà successivamente un potenziale problema da risolvere successivamente durante le addizioni e le sottrazioni. Poichè sono necessari solo quattro bit per codificare un digit bcd, ogni byte può codificare due digit BCD. Questo viene denominato "BCD impaccato".
Per esempio "00000000" sarà "00" in CD, "10011001" sarà "99".

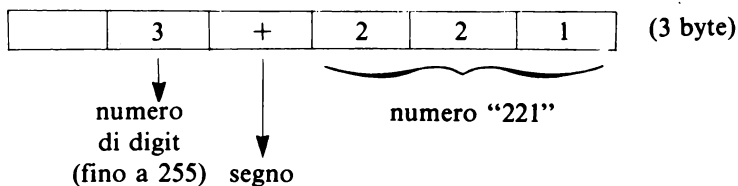
| CODICE | SIMBOLO BCD | CODICE | SIMBOLO BCD |
|---|---|---|---|
| 0000 | 0 | 1000 | 8 |
| 0001 | 1 | 1001 | 9 |
| 0010 | 2 | 1010 | non impiegato |
| 0011 | 3 | 1011 | non impiegato |
| 0100 | 4 | 1100 | non impiegato |
| 0101 | 5 | 1101 | non impiegato |
| 0110 | 6 | 1110 | non impiegato |
| 0111 | 7 | 1111 | non impiegato |

Figura 1-4: Tabella BCD

Un codice BCD si legge come segue:

**Esercizio 1.16:** *Qual'è la rappresentazione BCD di "29"? "91"?*

**Esercizio 1.17:** *"10100000" è una rappresentazione BCD valida? Perchè*

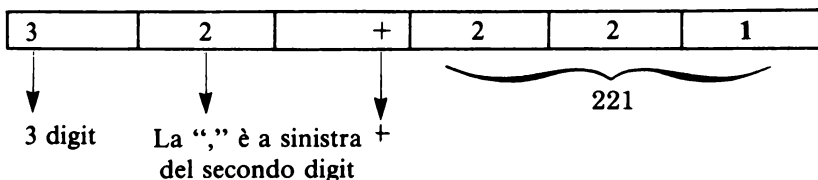
Per rappresentare tutti i digit BCD verranno utilizzati tutti i bit necessari. Tipicamente, all'inizio della rappresentazione, saranno utilizzati uno o più nibble per indicare il numero totale degli stessi, cioè il numero totale di digit BCD utilizzati. Un altro nibble o byte sarà utilizzato per indicare la posizione della virgola. Comunque le convenzioni possono essere diverse.

Questo è un esempio della rappresentazione di interi BCD multibyte.

Questo rappresenta + 221.
(Il segno può essere rappresentato da 0000 per + e da 0001 per —, per esempio).

**Esercizio 1.18:** *Si rappresenti "— 23123", impiegando la stessa convenzione. Si utilizzi inizialmente un formato BCD e quindi binario.*

**Esercizio 1.19:** *Si ricavi il corrispondente BCD di "222" e di "111", quindi del risultato di 222 × 111. (Si calcoli il risultato a mano e quindi si utilizzi la rappresentazione precedente).*

La rappresentazione BCD può essere utilizzata facilmente con i numeri decimali.

Per esempio + 2,21 può essere rappresentato da:

Il vantaggio della rappresentazione BCD è quello di ricavare dei risultati esatti in modo assoluto. Il suo svantaggio è di utilizzare una grande quantità di memoria e, conseguentemente, un procedimento aritmetico lento. Questo è accettabile solo nel settore della contabilità e, normalmente, non viene impiegato negli altri casi.

**Esecizio 1.20:** *Quanti bit sono richiesti per codificare "9999" in BCD? Ed in complemento a due?*

Abbiamo ora risolto il problema connesso con la rappresentazione dei numeri interi, degli interi con segno e dei numeri interi di valore assoluto

elevato. È già stato presentato un metodo possibile di rappresentazione dei numeri decimali, per mezzo della rappresentazione BCD. Si considererà ora il problema della rappresentazione dei numeri decimali in un formato di lunghezza fissa.

*Rappresentazione a Virgola Mobile*

Il principio di base è che i numeri decimali devono essere rappresentati con un formato fisso. Allo scopo di non sprecare bit, la rappresentazione *normalizzerà* tutti i numeri.

Per esempio "0,000123" spreca tre zeri a sinistra del numero che non hanno altro significato che di indicare la posizione del punto decimale. La normalizzazione di questo numero porge $0,123 \times 10^{-3}$. "0,123" è detta *mantissa normalizzata.* "— 3" è detto *esponente.* È stato normalizzato questo numero eliminando tutti gli zeri non significativi alla sua sinistra ed aggiustando l'esponente.

Consideriamo un altro esempio:

22,1 viene normalizzato come $0,221 \times 10^{-2}$
cioè $M \times 10^{E}$ dove M è la mantissa ed E l'esponente.

Si può vedere facilmente che un numero normalizzato è caratterizzato da una mantissa minore di uno e maggiore uguale a 0,1, in tutti i casi dove il numero considerato non è zero.
In altre parole questo può essere rappresentato matematicamente da:

$$0,1 \leqslant M < 1 \text{ oppure } 10^{-1} \leqslant M < 10^{0}$$

Analogamente nella rappresentazione binaria:

$$2^{-1} \leqslant M < 2^{0} \text{ (oppure } 0,5 \leqslant M < 1)$$

Dove M è il valore assoluto della mantissa (trascurando il segno).

Per esempio:

$$111,01 \text{ è normalizzato come: } 0,11101 \times 2^{3}.$$

La mantissa è 11101.

L'esponente è 3

È stato ora definito il principio della rappresentazione.

Si esamini il formato effettivo. La rappresentazione tipica a virgola mobile è mostrata in figura 1-5.

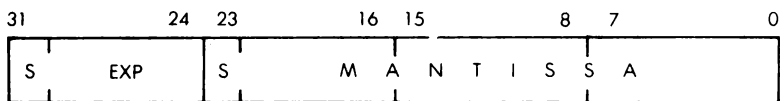



Figura 1-5: Rappresentazione tipica in virgola mobile.

Nella rappresentazione utilizzata in questo esempio sono impiegati quattro byte per un totale di 32 bit. Il primo byte a sinistra dell'illustrazione è impiegato per rappresentare l'esponente. Sia l'esponente che la mantissa saranno indicati mediante la rappresentazione in complemento a due.
Come risultato il massimo esponente sarà $2^{127}$ ed il più piccolo $2^{-128}$.

Tre byte sono impiegati per rappresentare la mantissa.
Poichè il primo bit della rappresentazione in complemento a due indica il segno, rimangono 23 bit per la rappresentazione della grandezza della mantissa.

**Esercizio 1.21:** *Quante cifre decimali possono essere rappresentate con i 23 bit della mantissa?*

Questo è solo uno degli esempi possibili di una rappresentazione a virgola mobile. È possibile utilizzare solo tre byte, oppure è possibile utilizzare più byte. La rappresentazione a quattro byte proposta sopra è proprio quella comune che rappresenta un ragionevole compromesso tra precisione, grandezza dei numeri, utilizzazione della memoria ed efficienza nelle operazioni aritmetiche.

Sono stati ora esaminati i problemi associati con la rappresentazione di numeri ed ora si conosce come rappresentarli in forma intera, con segno oppure in forma decimale. Si esaminerà ora come rappresentare internamente i dati alfanumerici.

**Rappresentazione dei Dati Alfanumerici**

La rappresentazione di dati alfanumerici, cioè caratteri, è completamente diretta: tutti i caratteri sono codificati in un codice di 8 bit. Solo due codici sono generalmente utilizzati nella parola del calcolatore, il Codice ASCII ed il Codice EBCDIC. ASCII significa "American Standard Code for Information Interchange" ed è universalmente utilizzato nella parola dei microprocessori. "EBCDIC" è una variante dell'ASCII utilizzata dalla IBM e perciò non è utilizzato nella parola di un microcalcolatore se non è realizzata un'interfaccia ad un terminale IBM.

Si esamini brevemente la codifica ASCII. Si possono codificare 26 lettere dell'alfabeto, maiuscole e minuscole, più 10 simboli numerici più circa 20 simboli addizionali speciali.
Questo può essere facilmente realizzato con 7 bit, che consentono 128 codici possibili. Tutti i caratteri sono perciò codificati in 7 bit: l'ottavo bit, quando utilizzato, è il *bit di parità.* La parità è una tecnica per verificare che i contenuti di una parola non siano stati accidentalmente cambiati. Viene contato il numero di uni della parole e l'ottavo bit è posto ad uno se il conteggio è dispari, rendendo così pari il numero totale. Questa è chiamata parità pari. Si può anche utilizzare la parità dispari, cioè scrivendo uno zero invece di un uno.

Esempio: si calcoli il bit di parità di "0010011" impiegando la parità pari. Il numero di uni è 3. Il bit di parità deve perciò essere un + 1, cosicchè il numero totale di uni è 4, cioè pari. Il risultato è 10010011, dove l'uno di testa è il bit di parità ed il numero 0010011 identifica il carattere.

La tabella dei codici ASCII a 7 bit appare in Fig. 1.6.

| HEX | MSD | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|
| LSD | BIT | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| 0 | 0000 | NUL | DLE | SPAZIO | 0 | @ | P | — | p |
| 1 | 0001 | SOH | DC1 | ! | 1 | A | Q | a | q |
| 2 | 0010 | STX | DC2 | " | 2 | B | R | b | r |
| 3 | 0011 | ETX | DC3 | # | 3 | C | S | c | s |
| 4 | 0100 | EOT | DC4 | $ | 4 | D | T | d | t |
| 5 | 0101 | ENQ | NAK | % | 5 | E | U | e | u |
| 6 | 0110 | ACK | SYN | & | 6 | F | V | f | v |
| 7 | 0111 | BEL | ETB | ' | 7 | G | W | g | w |
| 8 | 1000 | BS | CAN | ( | 8 | H | X | h | x |
| 9 | 1001 | HT | EM | ) | 9 | I | Y | i | y |
| A | 1010 | LF | SUB | * | : | J | Z | j | z |
| B | 1011 | VT | ESC | + | ; | K | [ | k | { |
| C | 1100 | FF | FS | ' | < | L | \ | l | - - |
| D | 1101 | CR | GS | — | = | M | ] | m | } |
| E | 1110 | SO | RS | . | > | N | Λ | n | ~ |
| F | 1111 | SI | US | / | ? | O | ← | o | DEL |

**Figura 1-6: Tabella di conversione (vedere Appendice E per le abbreviazioni)**

In pratica essa viene utilizzata sia direttamente, cioè senza parità, aggiungendo uno 0 nella posizione di estrema sinistra, sia con la parità, aggiungendo il bit opportuno sulla sinistra.

**Esercizio 1.22:** *Si calcoli la rappresentazione ad 8 bit delle cifre da "0" a "9", utilizzando la parità pari. (Questo codice verrà utilizzato nell'esempio di applicazione del Capitolo 8).*

**Esercizio 1.23:** *Si esegua lo stesso procedimento sulle lettere dalla "A" alla "F".*

In settori specifici, come quello delle telecomunicazioni, possono essere utilizzati altri metodi di codifica, come i codici a correzione di errore. Comunque essi esulano dallo scopo di questo libro.
Si è imparato come rappresentare sia il programma che i dati all'interno del calcolatore. Si esamineranno ora le possibili rappresentazioni esterne.

## *RAPPRESENTAZIONE ESTERNA DELL'INFORMAZIONE*

La rappresentazione esterna fa riferimento al modo in cui l'informazione è presentata all'*utente,* cioè generalmente al programmatore. L'informazione può essere rappresentata esternamente essenzialmente in tre formati.

### *1. Binario*

Si è visto che l'informazione è immagazzinata internamente in *bit* (sequenze di zeri ed uni). È talvolta desiderabile mostrare questa informazione interna direttamente nel suo formato binario e questa è chiamata *rappresentazione binaria.*
Un semplice esempio è fornito dalle luci del pannello frontale di un microcalcolatore (se esso ha un pannello frontale). Nel caso di un microprocessore ad 8 bit, un pannello frontale sarà tipicamente fornito di 8 LED per mostrare i contenuti di qualsiasi registro.
Un LED illuminato indica un uno, un LED che non è illuminato indica uno zero. Tale rappresentazione binaria può essere necessaria per un debugging accurato di un programma complesso, specialmente se esso coinvolge operazioni di ingresso-uscita, ma è naturalmente impraticabile a livello umano. Si sono evolute rappresentazioni più convenienti.

### *2. Ottale ed Esadecimale*

L'"Ottale" e l'"Esadecimale" codificano rispettivamente tre e quattro

bit binari in un'unico simbolo. Nel sistema ottale, qualsiasi rappresentazione di tre bit binari è rappresentata da un numero tra 0 e 7.

"Ottale" è un formato che impiega tre bit, nel quale ogni combinazione di tre bit è rappresentata da un simbolo tra 0 e 7:

| binario | ottale |
|---|---|
| 000 | 0 |
| 001 | 1 |
| 010 | 2 |
| 011 | 3 |
| 100 | 4 |
| 101 | 5 |
| 110 | 6 |
| 111 | 7 |

Figura 1.7: Simboli ottali

Per esempio, "00 100 100" in binario, nella rappresentazione ottale diventerebbe: ▼ ▼ ▼
0 4 4

cioè "044" in ottale.

Un altro esempio: 11 111 111 è:
▼ ▼ ▼
3 7 7

ovvero "377" in ottale.

Inversamente in numero ottale "211" rappresenta:

010 001 001

o "10001001" binario.

Tradizionalmente l'ottale è stato utilizzato nei computer più vecchi che interamente utilizzavano vari numeri di bit da 8 a circa 64.

Più recentemente, col dominio dei microprocessori ad 8 bit, il formato ad 8 bit è divenuto quello convenzionale ed è utilizzata un'altra rappresentazione che è più pratica. Questa è l'*esadecimale.*

Nella rappresentazione esadecimale un gruppo di quattro bit è codificato come digit esadecimale. I digit esadecimali sono i numeri da 0 a 9, seguiti dalle lettere A, B, C, D, E, F. Per esempio "0000" è rappresentato da "0", "0001" è rappresentato da "1" ed "1111" è rappresentato dalla lettera "F". (Vedere Fig. 1-8).

| DECIMALE | BINARIO | ESADEC. | OTTALE |
|---|---|---|---|
| 0 | 0000 | 0 | 0 |
| 1 | 0001 | 1 | 1 |
| 2 | 0010 | 2 | 2 |
| 3 | 0011 | 3 | 3 |
| 4 | 0100 | 4 | 4 |
| 5 | 0101 | 5 | 5 |
| 6 | 0110 | 6 | 6 |
| 7 | 0111 | 7 | 7 |
| 8 | 1000 | 8 | 10 |
| 9 | 1001 | 9 | 11 |
| 10 | 1010 | A | 12 |
| 11 | 1011 | B | 13 |
| 12 | 1100 | C | 14 |
| 13 | 1101 | D | 15 |
| 14 | 1110 | E | 16 |
| 15 | 1111 | F | 17 |

Figura 1.8: Codici esadecimali

Esempio: $\underbrace{1010}_{A}\ \underbrace{0001}_{1}$ in binario, è rappresentato da: A 1 in esadecimale

**Esercizio 1.24:** *Qual'è la rappresentazione esadecimale di "10101010"?*

**Esercizio 1.25:** *Inversamente qual'è il binario equivalente di "FA" esadecimale?*

**Esercizio 1.26:** *Qual'è l'ottale di "01000001"?*

L'esadecimale offre il vantaggio di codificare i bit in soli due digit. Questo è più facile da visualizzare o memorizzare e più veloce da rappresentare. Perciò, su tutti i nuovi microcalcolatori, l'esadecimale è il metodo preferito di rappresentazione di gruppi di bit.

Naturalmente, ogni volta che l'informazione presente nella memoria ha un significato, cioè rappresenta un testo, o numeri, l'esadecimale non è conveniente rispetto ad altri per rappresentare il significato di questa informazione, per l'impiego umano diretto.
In questi casi si potrebbe utilizzare un terzo metodo.

*3. Rappresentazione Simbolica*

La *rappresentazione simbolica* conduce alla rappresentazione esterna dell'informazione in forma effettiva. Per esempio i numeri decimali sono rappresentati come numeri decimali e non come sequenze di simboli esadecimali o bit. Analogamente il testo è rappresentato come tale. Naturalmente la rappresentazione simbolica è molto più pratica all'utente. Essa è impiegata ogni volta che è disponibile un appropriato dispositivo display, come un display CRT oppure una stampante. Sfortunatamente, nei sistemi più piccoli come i microcomputer su scheda singola, non è economico fornire tali display e l'utente è limitato alla comunicazione in esadecimale col microcalcolatore.

## Sommario delle Rappresentazioni Esterne

La rappresentazione simbolica dell'informazione è la più auspicabile poichè è la più naturale per l'utente umano. Comunque essa richiede un'interfaccia dispendiosa sotto forma di una tastiera alfanumerica più una stampante oppure un display CRT. Per questa ragione essa non può essere disponibile sui sistemi meno dispendiosi. Un tipo alternativo di rappresentazione viene quindi utilizzato ed in questo caso l'esadecimale è la rappresentazione dominante.
Solo in rari casi di correlazione con un accurato debugging a livello hardware o software viene utilizzata la rappresentazione binaria. Il *Binario* mostra direttamente i contenuti dei registri o della memoria in formato binario.

Si è ora imparato come rappresentare l'informazione internamente ed esternamente. Si esaminerà ora il microprocessore effettivo che manipolerà quest'informazione.

**Ulteriori Esercizi**

**Esercizio 1.27:** *Qual'è il vantaggio del complemento a due rispetto alle altre rappresentazioni?*

**Esercizio 1.28:** *Come si potrebbe rappresentare "1024" direttamente in binario? In binario con segno? In complemento a due?*

**Esercizio 1.29:** *Cos'è il bit V? Il programmatore dovrebbe verificarlo dopo un'addizione o sottrazione?*

**Esercizio 1.30:** *Si calcolino i complementi a due di "+ 16", "+ 17", "+ 18", "– 17", "– 18".*

**Esercizio 1.31:** *Si mostri la rappresentazione esadecimale del testo seguente, che è stato memorizzato internamente nel formato ASCII senza parità: = "MESSAGE".*

CAPITOLO 2

# ORGANIZZAZIONE HARDWARE DEL 6502

## INTRODUZIONE

Per programmare a livello elementare non è necessario comprendere in dettaglio la struttura interna del processore che si sta utilizzando. Comunque, per una programmazione efficiente, tale comprensione è richiesta. Lo scopo di questo capitolo è di presentare i concetti hardware di base necessari per la comprensione del funzionamento del sistema 6502. Il sistema completo del microcalcolatore comprende non solo l'unità del microprocessore (cioè il 6502) ma anche altri componenti. Questo capitolo presenta il 6502 vero e proprio; invece gli altri dispositivi (principalmente d'ingresso/uscita) saranno presentati in un capitolo separato. (Capitolo 7).

Si analizzerà di seguito l'architettura di base del sistema microcalcolatore, quindi si studierà in dettaglio l'organizzazione interna del 6502. Si esamineranno in particolare, i vari registri. Si studierà poi l'esecuzione del programma ed i meccanismi sequenziali. Da un punto di vista hardware questo capitolo è solo una presentazione semplificata. Il lettore interessato ad ottenere una comprensione dettagliata si riferisca al nostro libro ("Microprocessori", dello stesso autore).

## ARCHITETTURA DEL SISTEMA

L'architettura del sistema microcalcolatore è mostrata in Figura 2.1. L'unità del microprocessore (MPU), che in questo caso sarà un 6502, appare a sinistra dell'illustrazione. Essa realizza le funzioni di una *unità di elaborazione centrale* (CPU) all'interno di un chip: essa comprende un'*unità aritmetico-logica* (ALU), più i suoi registri interni, ed una *unità di controllo* (CU) avente il compito di sequenziare il sistema. Il suo funzionamento sarà spiegato in questo capitolo.

La MPU origina tre *buss:* un *bus dati* bidirezionale ad 8 bit, che compare alla sommità dell'illustrazione, un *bus indirizzi* bidirezionale a 16 bit ed un *bus di controllo* che appaiono in basso nell'illustrazione. Si descriverà la funzione di ciascuno di questi bus.

Il *bus dati* trasferisce i dati che devono essere scambiati dai vari elementi del sistema. Tipicamente esso trasferirà i dati dalla memoria alla MPU, oppure dalla MPU alla memoria, oppure dalla MPU ad un chip d'ingresso/uscita. (Un chip d'ingresso/uscita è un componente che si incarica di comunicare con un dispositivo esterno).

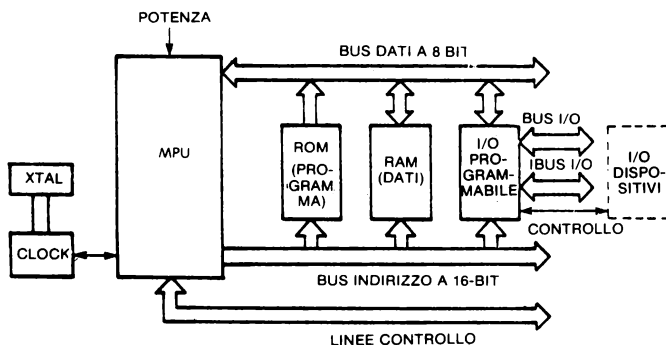



Figura 2.1: Architettura di un sistema a microprocessore convenzionale

Il *bus indirizzi* trasferisce un *indirizzo* generato dalla MPU, che selezionerà un registro interno di uno dei chip connessi al sistema. Questo indirizzo specifica la sorgente, ovvero la destinazione dei dati che transiteranno lungo il bus dati.

Il *bus controllo* trasferisce i vari segnali di sincronizzazione richiesti dal sistema.

Descritto lo scopo dei bus, si considerano ora le connessioni dei componenti addizionali richiesti da un sistema completo.

Ogni MPU richiede un preciso timing di riferimento che è fornito da un *clock* e da un quarzo. Nella maggior parte dei microprocessori "più vecchi", l'oscillatore del clock è esterno alla MPU e richiede un ulteriore chip. Nei microprocessori più recenti, l'oscillatore del clock è normalmente incorporato all'interno della MPU. Il cristallo di quarzo, comunque, a causa del suo volume, è sempre esterno al sistema. Il cristallo ed il clock compaiono a sinistra del blocco MPU nell'illustrazione.

Si rivolga ora l'attenzione ad altri elementi del sistema.
Andando da sinistra a destra nell'illustrazione, si distingue: il blocco ROM è la *memoria a sola lettura* e contiene il *programma* per il sistema. Il vantaggio della memoria ROM è che i suoi contenuti sono permanenti e

non scompaiono ogni volta che il sistema viene spento. La ROM perciò contiene sempre un *bootstrap* ovvero un programma *monitor* (la loro funzione sarà spiegata in seguito) che consente il funzionamento iniziale del sistema. Nei controlli di processo quasi tutti i programmi risiedono su ROM poichè essi probabilmente non saranno mai cambiati. In tali casi l'utente industriale deve proteggere il sistema contro interruzioni dell'alimentazione: i programmi possono non essere volatili. Essi devono essere su ROM.

Comunque, nelle condizioni di hobby, ovvero in uno sviluppo di programma (quando il programmatore prova il suo programma), la maggior parte dei programmi risiederanno su RAM cosicchè essi possano essere facilmente cambiati. In seguito essi possono rimanere su RAM oppure essere trasferiti su ROM, se richiesto. Comunque la RAM è volatile. I suoi contenuti vanno persi se viene a mancare l'alimentazione.

La RAM (Random-Access-Memory) è la memoria di lettura/scrittura del sistema. Nel caso di sistema di controllo la quantità di RAM sarà tipicamente piccola (solo per i dati).

D'altra parte, nel caso di sviluppo di programma, la quantità di RAM sarà grande e conterrà i programmi più il software di sviluppo. Tutti i contenuti RAM devono essere caricati prima dell'impiego da un dispositivo esterno.

Infine il sistema conterrà uno o più chip di interfaccia. Quello usato più spesso è il *"PIO"* ovvero chip d'Ingresso/Uscita Parallelo. È quello mostrato in figura. Questo PIO come tutti gli altri chip del sistema, è collegato a tutti e tre i bus e fornisce almeno due *porte* a 16 bit per comunicare col mondo esterno. Per maggiori dettagli, su come effettivamente lavora un PIO, ci si riferisca al libro "Microprocessori" ovvero anche, specificamente per il sistema 6502, si faccia riferimento al Capitolo 7 (dispositivi di Ingresso-Uscita).

Tutti questi chip sono connessi a tutti i tre bus, compreso il bus controllo. Comunque, per chiarire l'illustrazione, le connessioni tra bus controllo e questi vari chip non sono mostrate sul diagramma.

I moduli funzionali descritti non necessariamente risiedono su un unico chip LSI. Infatti si useranno chip di *combinazione* che comprendano sia un PIO ed una limitata quantità di ROM o RAM. Per maggiori dettagli si faccia riferimento al Capitolo 7.

Ancora più componenti saranno richiesti per costruire un sistema reale. In particolare i bus normalmente richiedono dei *buffer.* Anche la *logica di decodifica* può essere utilizzata per i chip di memoria RAM ed infine qualche segnale può richiedere di essere amplificato per mezzo di *driver.*

Questi circuiti ausiliari non verranno descritti perchè non sono importanti dal punto di vista della programmazione. Il lettore interessato in particolare all'assembly ed alle tecniche di realizzazione di interfacce viene indirizzato al libro: "Tecniche di Interfacciamento dei Microprocessori".

## ORGANIZZAZIONE INTERNA DEL 6502

Un diagramma semplificato dell'organizzazione interna del 6502 appare in Figura 2.2.

L'unità aritmetico logica (ALU) compare sulla destra dell'istruzione. Essa può essere facilmente riconosciuta dalla caratteristica sagoma a "V". La funzione dell'ALU è l'esecuzione di operazioni aritmetiche e logiche sui dati che la alimentano attraverso le sue due porte d'ingresso. Queste due porte d'ingresso della ALU sono rispettivamente l'"Ingresso sinistro" e l'"Ingresso destro". Essi corrispondono alle due estremità più alte della sagoma a "V". Dopo l'esecuzione di un'operazione aritmetica come una addizione o sottrazione, l'ALU fa uscire i suoi contenuti dal fondo dell'illustrazione.

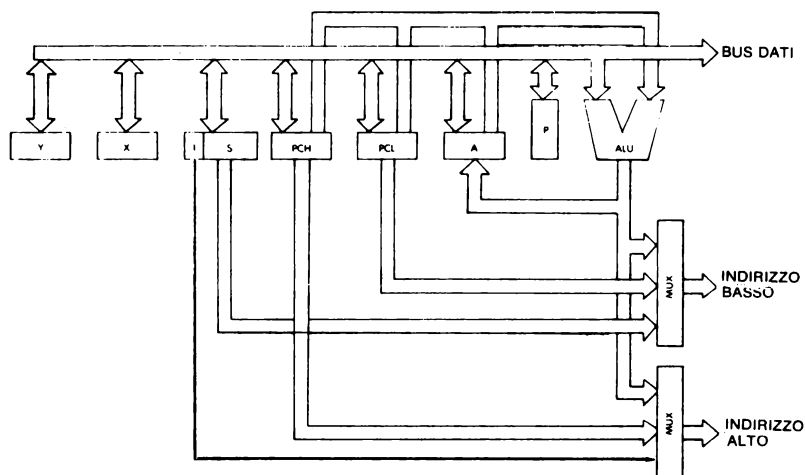



Figura 2.2: Organizzazione interna del 6502

L'ALU è equipaggiata di uno speciale registro, *l'accumulatore* (A). L'accumulatore è sull'ingresso sinistro. L'ALU fa riferimento automaticamente a questo accumulatore come ad uno degli ingressi. (Comunque

esiste anche un by-pass). Questo è un progetto classico basato sull'accumulatore. Nelle operazioni aritmetiche e logiche, uno degli operandi sarà nell'accumulatore e l'altro si troverà tipicamente in una locazione di memoria. Il risultato sarà depositato nell'accumulatore. Il riferimento dell'accumulatore come sorgente e destinazione dei dati è la ragione del suo nome: esso accumula i risultati. Il vantaggio di questo approccio basato sull'accumulatore è la possibilità di impiegare istruzioni molto corte, appena un singolo byte (8 bit) per specificare il codice operativo od "opcode", cioè la natura dell'operazione da eseguire. Se uno degli operandi deve essere prelevato da uno degli altri registri (diversi dall'accumulatore), sarebbe necessario utilizzare ulteriori bit per indicare quale registro all'interno dell'istruzione. Perciò l'architettura dell'accumulatore si risolve in una maggiore velocità di esecuzione. Lo svantaggio è che l'accumulatore deve sempre essere caricato con i dati richiesti prima della sua utilizzazione. Questo può risolversi in qualche punto inefficiente.

Si ritorni all'illustrazione. Di fianco alla ALU, alla sua sinistra, appare uno speciale registro ad 8 bit, i *flag di stato* del processore (P). Ciascuno di questi bit, realizzato fisicamente da un flip-flop all'interno del registro è utilizzato per denotare una condizione speciale. La funzione dei vari bit di stato sarà spiegata successivamente durante gli esempi di programmazione che saranno presentati nel capitolo successivo e saranno completamente descritti nel Çapitolo 4 che presenta il set di istruzini completo. Come esempio tre di tali flag di stato sono i bit N, Z e C.

N sta per "negativo". Esso è il bit 7 (cioè il più a sinistra) del registro P. Ogni volta che questo bit è al livello logico uno indica che il risultato dell'operazione eseguita dalla ALU è negativo.

Il bit Z sta per zero. Ogni volta che questo bit (posizione di bit 1) è ad uno, si denota che è stato ottenuto un risultato zero.

Il bit C, nella posizione più a destra, (posizione 0), è un bit *carry,* cioè di riporto. Ogni volta che sono sommati due numeri di 8 bit ed il risultato non può essere contenuto in 8 bit, il bit C è il nono bit del risultato. Il carry è usato in modo estensivo durante le operazioni aritmetiche.

Questi bit di stato sono controllati automaticamente dalle varie istruzioni. Una lista completa delle istruzioni ed il modo in cui esse influenzano i bit di stato del sistema appare nell'Appendice a come pure al Capitolo 4. Questi bit saranno utilizzati dal programmatore per varie verifiche speciali o condizioni eccezionali, oppure per verificare velocemente alcuni risultati errati. Per esempio la verifica del bit Z può essere associata con istruzioni speciali e dirà immediatamente se il risultato di una prcedente operazione era 0 oppure no. Tutte le decisioni in un

programma in linguaggio assembly, cioè in tutti i programmi che saranno sviluppati in questo libro, saranno basati sulla verifica di bit. Questi bit saranno sia i bit che saranno letti dal mondo esterno, oppure i bit di stato della ALU. È perciò molto importante capire la funzione e l'uso di tutti i bit di stato del sistema. La ALU è dotata di un registro di stato contenente questi bit. Tutti gli altri chip di ingresso/uscita del sistema saranno anch'essi equipaggiati con bit di stato. Questo sarà studiato al Capitolo 7.

Ci si muova ora verso sinistra dalla ALU nell'illustrazione 2.2. I rettangoli orizzontali rappresentano i registri interni del 6502.

PC è il *contatore di programma.* È un registro a 16 bit ed è realizzato fisicamente come due registri ad 8 bit: PCL e PCH, PCL costituisce la metà di basso livello del contatore di programma, cioè i bit da 0 a 7. PCH costituisce la parte ad alto livello del contatore di programma cioè i bit da 8 a 15. Il contatore di programma è un registro a 16 bit che contiene l'indirizzo dell'istruzione successiva da eseguire. Ogni calcolatore è equipaggiato con un contatore di programma in modo da conoscere quale istruzione deve essere successivamente eseguita. Si analizza brevemente il meccanismo di accesso alla memoria in modo da mostrare il ruolo del contatore di programma.

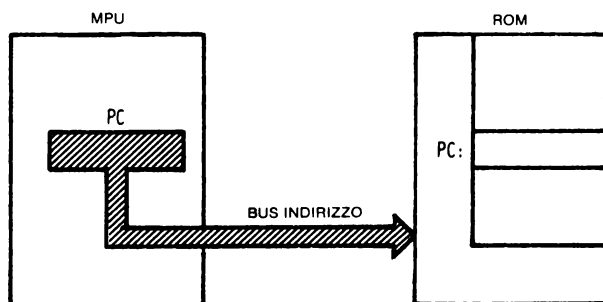



Figura 2.3: Prelievo di un'istruzione dalla memoria

## IL CICLO DI ESECUZIONE DI UN'ISTRUZIONE

Si faccia riferimento alla Figura 2-3. L'unità microprocessore appare a sinistra e la memoria a destra. Il chip di memoria può essere una ROM oppure una RAM oppure qualsiasi altro chip che svolga le funzioni di memoria. La memoria è utilizzata per immagazzinare istruzioni e dati.

Ora si seguirà il prelievo di un'istruzione della memoria per illustrare il ruolo del contatore di programma. Si supponga che il contatore di programma abbia un certo contenuto valido. Esso conserva così un indirizzo a 16 bit che è l'indirizzo dell'istruzione successiva da prelevare dalla memoria. Qualsiasi processore procede in tre cicli:

1 — Prelievo dell'istruzione successiva
2 — Decodifica dell'istruzione
3 — Esecuzione dell'istruzione

*Prelievo (fetch)*

Si segue la sequenza. Nel primo ciclo i contenuti del contatore di programma sono depositati sul bus indirizzo e portati alla memoria (sul bus indirizzo stesso). Contemporaneamente un segnale di lettura può essere emesso sul bus controllo del sistema, se richiesto. La memoria riceverà l'indirizzo. Questo indirizzo è utilizzato per specificare una locazione all'interno della memoria. Dopo la ricezione del segnale di lettura la memoria decodificherà l'indirizzo ricevuto, per mezzo di decodificatori interni, e selezionerà la locazione specificata dall'indirizzo. Alcune centinaia di nanosecondi più tardi, la memoria depositerà i dati ad 8 bit, corrispondenti all'indirizzo specificato, sul suo bus dati. Questa parola ad 8 bit è l'istruzione che si vuole prelevare. Nell'illustrazione precedente quest'istruzione sarà depositata sulla sommità del bus dati.

Si riassuma brevemente la sequenza: i contenuti del contatore di programma sono inviati sul bus indirizzo. Viene generato un segnale di lettura. La memoria entra in funzione. Circa 300 nanosecondi più tardi, l'istruzione dell'indirizzo specificato è depositata sul bus dati. Il microprocessore quindi legge il bus dati e depone i suoi contenuti in un registro interno specializzato, il registro IR. Il registro IR è il *registro di istruzione.* Esso è largo 8 bit ed è utilizzato per contenere l'istruzione appena prelevata dalla memoria. Il ciclo di prelievo è così completato. Gli 8 bit dell'istruzione si trovano ora fisicamente nello speciale registro interno del 6502, il registro IR. Questo registro IR appare a sinistra nella Figura 2-4.

*Decodifica ed Esecuzione*

Una volta che l'istruzione è contenuta nell'IR, l'unità di controllo del microprocessore decodificherà i contenuti e sarà in grado di generare la sequenza corretta dei segnali interni ed esterni per l'esecuzione dell'istruzione specificata. C'è perciò un breve ritardo di decodifica seguito da

una fase di esecuzione, la cui larghezza dipende dalla natura dell'istruzione specificata. Alcune istruzioni saranno eseguite interamente all'interno della MPU. Altre istruzioni preleveranno o depositeranno dati dalla o nella memoria. Questa è la ragione per cui le diverse istruzioni del 6502 richiedono diversi tempi di esecuzione. La durata è espressa come numero di cicli (di clock). Si faccia riferimento all'Appendice per il numero di cicli richiesti da ciascuna istruzione. Tipicamente il 6502 utilizza un clock di 1 MHz. La lunghezza di ogni ciclo è perciò un microsecondo. Poichè possono essere utilizzate varie velocità di clock con componenti diversi, la velocità di esecuzione è normalmente espressa in numero di cicli piuttosto che in numero di nanosecondi.

Si noterà anche che sulla parte più a sinistra dell'illustrazione compare un oscillatore interno al 6502. Questo è il clock che è interno nel caso del 6502.

**Prelievo dell'istruzione successiva**

È stata appena descritta l'utilizzazione del contatore di programma e come un'istruzione può essere prelevata dalla memoria. Durante l'esecuzione di un programma, le istruzioni sono prelevate *in sequenza* dalla memoria. Occorre perciò fornire un meccanismo automatico per prelevare le istruzioni in modo sequenziale. Questo compito è eseguito da un semplice incrementatore connesso al contatore di programma. Questo è illustrato in Figura 2-4. Ogni volta che i contenuti del contatore di programma (in basso nell'illustrazione) sono posizionati sul bus indirizzo, i contenuti dello stesso contatore saranno incrementati e riscritti nel contatore stesso. Per esempio, se il contatore di programma contenesse il valore 0, il valore 0 uscirebbe sul bus indirizzo. Allora i contenuti del contatore di programma sarebbero incrementati ed il valore 1 sarebbe riscritto nel contatore stesso. In questo modo la volta successiva che il contatore di programma viene utilizzato, sarà prelevata l'istruzione all'indirizzo 1. Si è così realizzato un meccanismo automatico per sequenziare le istruzioni.

Si deve sottolineare che la precedente descrizione è semplificata. In realtà alcune istruzioni possono essere lunghe 2 od anche 3 byte cosicchè i byte successivi saranno prelevati in questo modo dalla memoria. Comunque il meccanismo è identico. Il contatore di programma è utilizzato per prelevare byte successivi di un'istruzione allo stesso modo del prelievo di istruzioni successive. Il contatore di programma, assieme al suo incrementatore, fornisce un meccanismo automatico per il puntamento alle locazioni di memoria successive.

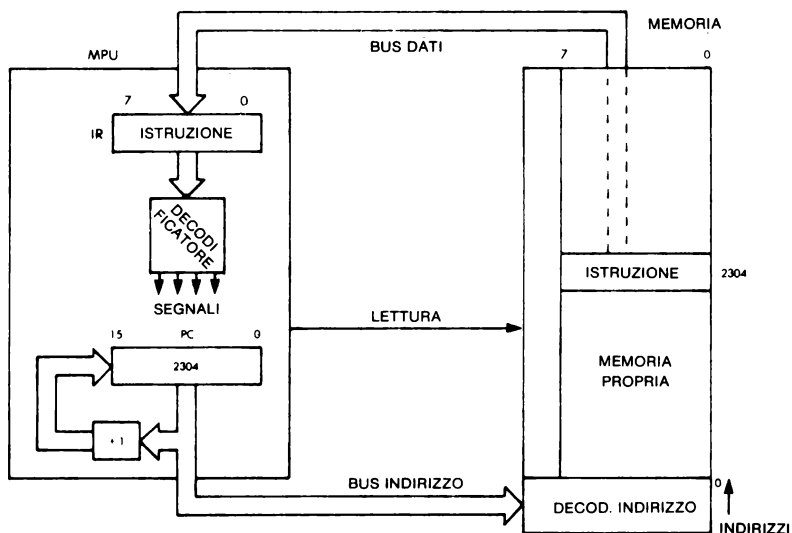



Figura 2.4: Sequenza automatica

## Altri Registri del 6502

Un'ultima area della Figura 2-2 non è ancora stata spiegata. Questa comprende l'insieme dei tre registri indicati X, Y ed S. I registri X ed Y sono chiamati registri indice. Essi sono larghi 8 bit. Essi possono essere utilizzati per contenere dati su cui opererà il programma. Comunque essi sono normalmente utilizzati come registri indice.

Il ruolo dei registri indice sarà descritto al Capitolo 5 sulle tecniche di indirizzamento. Brevemente, i contenuti di questi due registri indice possono essere sommati in diversi modi a qualsiasi registro specificato all'interno del sistema per fornire una scelta automatica. Questa è una caratteristica importante per recuperare in modo efficiente i dati quando sono immagazzinati in tabelle. Questi due registri non sono completamente simmetrici ed il loro ruolo sarà differenziato nel capitolo sulle tecniche di indirizzamento.

Il registro dello stack S è utilizzato per contenere un puntatore alla sommità dell'area dello stack all'interno della memoria.
Si introdurrà ora il concetto formale di uno stack.

## LO STACK

Uno stack è formalmente chiamato una struttura LIFO (last-in, first-out). Uno stack è un insieme di registri, o locazioni di memoria, allocati per questa struttura dati. La caratteristica essenziale di questa struttura è che si tratta di una struttura *cronologica.* Il primo elemento introdotto nello stack è sempre in fondo allo stack. L'ultimo elemento depositato nello stack è sempre alla sommità dello stack. Si può tracciare un'analogia con i piatti su un contatore di un ristorante.

C'è un foro sul contatore con una molla sul fondo. I piatti sono infilati sopra il foro. Con questa organizzazione è sicuro che il piatto introdotto per primo nello stack è sempre in fondo. Quello che è stato posizionato più recentemente sullo stack è quello alla sommità. Questo esempio illustra anche un'altra caratteristica dello stack. Nell'impiego normale uno stack è accessibile solo attraverso due istruzioni "push" e "pop" (o "pull"). L'operazione *push* (spinge) fa depositare un elemento alla sommità dello stack. L'operazione *pull* (estrae) consiste nella rimozione di un elemento dallo stack. In patica, nel caso di un microprocessore, è *l'accumulatore* che sarà depositato alla sommità dello stack. L'operazione *pop* conduce ad un trasferimento dell'elemento di sommità dello stack nell'accumulatore. Possono esistere altre istruzioni specializzate per trasferire la sommità dello stack tra altri registri specializzati, come il registro di stato.

È richiesta la disponibilità di uno stack per realizzare tre possibilità di programmazione all'interno del sistema calcolatore: subroutine, interrupt ed immagazzinamento temporaneo di dati. Il ruolo dello stack durante le subroutine sarà spiegato nel Capitolo 3 (Tecniche di Programmazione di Base).
Il ruolo dello stack durante gli interrupt sarà spiegato al Capitolo 6 (Tecniche di Ingresso/Uscita). Infine il ruolo dello stack nella conservazione di dati ad alta velocità sarà spiegato nel corso di programmi specifici di applicazione.

A questo punto si assumerà semplicemente che lo stack è una caratteristica richiesta in qualsiasi sistema calcolatore.
Uno stack può essere realizzato in due modi:

1. Un numero fisso di registri può essere fornito all'interno del microprocessore stesso. Questo è uno "stack hardware". Questo ha il vantaggio di un'alta velocità. Comunque ha lo svantaggio di un limitato numero di registri.

2. La maggior parte dei microprocessori general-purpose scelgono un altro approccio, lo stack software, in modo da non restringere lo stack ad

un numero molto piccolo di registri. Questo è l'approccio scelto nel 6502. Nell'approccio software un registro orientato all'interno del microprocessore, il registro S in questo caso, immagazzina il puntatore dello stack, cioè l'indirizzo dell'elemento di sommità dello stack più uno). Lo stack è poi realizzato come un'area di memoria. Il puntatore dello stack richiederà perciò 16 bit per poter puntare ovunque nella memoria.

Comunque, nel caso del 6502, il puntatore dello stack è ristretto ad 8 bit. Esso comprende un nono bit, nella posizione più a sinistra, sempre posto ad 1. In altre parole l'area riservata allo stack nel caso del 6502 va dall'indirizzo 256 all'indirizzo 511. In binario questo è da "100000000" a "111111111". Lo stack inizia sempre all'indirizzo 111111111 e può avere fino a 255 parole. Questo può essere visto come una limitazione del 6502 e sarà discusso successivamente in questo libro. Nel 6502 lo stack è all'indirizzo più alto e si sviluppa "all'indietro": il puntatore dello stack è decrementato da un'istruzione PUSH.

Per utilizzare lo stack il programmatore inizializzerà semplicemente il registro S. Il resto è automatico.

Lo stack viene considerato risiedere nella *pagina 1* della memoria. Si introdurrà ora il concetto di *impaginazione.*

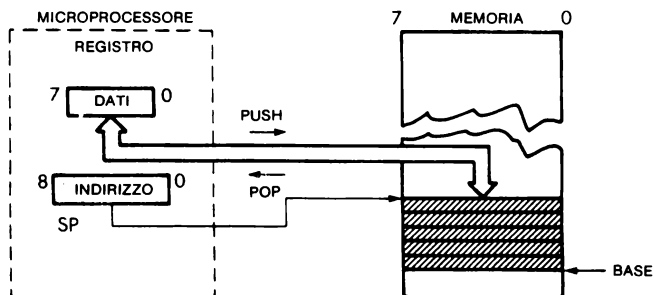



Figura 2.5: Le 2 istruzioni di manipolazione dello stack

## IL CONCETTO DI IMPAGINAZIONE

Il microprocessore 6502 è equipaggiato con un bus indirizzo a 16 bit. Si possono utilizzare 16 bit binari per creare fino a $2^{16} = 64$ K combinazioni (1 K è uguale a 1024). A causa delle caratteristiche di indirizzamento del 6502 che saranno presentate al Capitolo 5, è conveniente la

partizione della memoria in *pagine* logiche. Una pagina è semplicemente un blocco di 256 parole. Così le locazioni di memoria da 0 a 255 sono la pagina 0 della memoria. Esse saranno utilizzate per l'indirizzamento "Pagina zero". La pagina 1 della memoria comprende le locazioni di memoria da 256 a 511. È stato appena stabilito che la pagina 1 è normalmente riservata all'area stack. Tutte le altre pagine del sistema non sono coinvolte dal progetto e possono essere utilizzate liberamente. Nel caso del 6502 è importante ricordare l'organizzazione a pagina della memoria. Ogni volta che viene attraversata la frontiera di una pagina si introduce spesso un ulteriore ciclo di ritardo nell'esecuzione di un'istruzione.

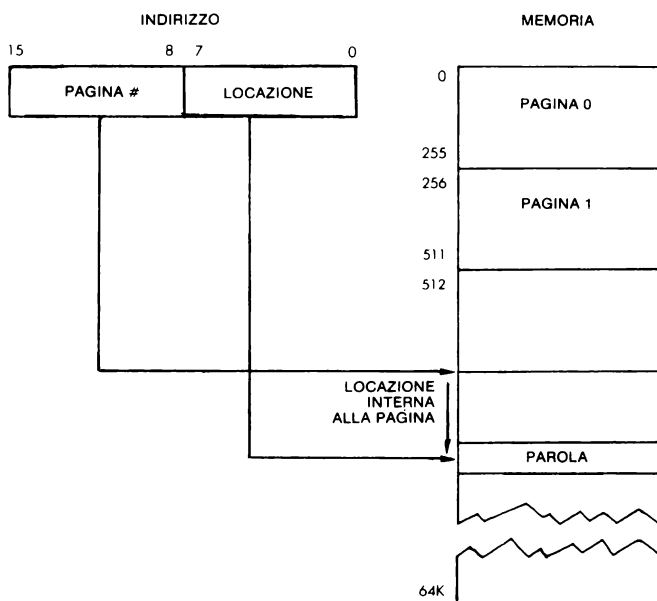



Figura 2.6: Il concetto di impaginazione

## IL CHIP 6502

Per completare la nostra descrizione del diagramma, il bus dati, riportato nella parte superiore della Figura 2-2, rappresenta il bus dati esterno. Esso sarà utilizzato per comunicare con i dispositivi esterni ed in particolare la memoria.

A0-7 ed A8-15 rappresentano rispettivamente le parti di basso ed alto ordine del bus indirizzo creato dal 6502.

Per completezza si presenta di seguito l'effettiva disposizione dei pin del microprocessore 6502. Non è necessario leggere questo per capire il resto di questo libro. Comunque se si intende connettere il dispositivo ad un sistema questa descrizione sarà preziosa.

L'effettiva disposizione dei pin del 6502 appare in Figura 2-7. Il bus dati è indicato con la label DB0-7 ed è facilmente riconoscibile sulla destra dell'illustrazione. Il bus indirizzo è indicato con la label A0-11 ed A12-15. Esso comprende i pin dal 9 al 20 a sinistra del chip ed i pin 22 a destra.

I segnali rimanenti sono l'alimentazione ed i segnali di controllo.

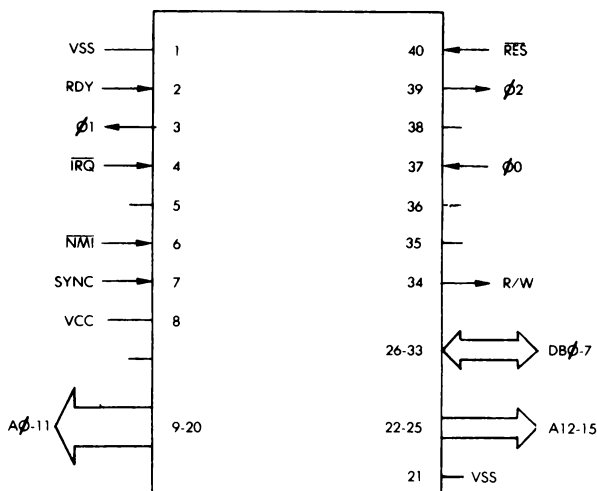



Figura 2.7: Disposizione dei pin del 6502

*I segnali di controllo*

—R/W: è la linea di controllo di LETTURA/SCRITTURA nella direzione del trasferimento dei dati sul bus dati.

—IRQ ed NMI sono la "Richiesta di Interrupt" e l'Interrupt Non Mascherabile. Queste sono due linee di interrupt e saranno utilizzate al Capitolo 7.

—SYNC è un segnale che indica il prelievo di un codice operativo dal mondo esterno.

—RDY è normalmente utilizzato per il sincronismo con una memoria lenta; esso arresterà il processore.
—SO comanda il flag di overflow. Normalmente non viene utilizzato.
—$\emptyset_0$, $\emptyset_1$ e $\emptyset_2$ sono i segnali di clock.
—RES è il RESET, impiegato per inizializzare.
—Vss e Vcc sono le alimentazioni (5V).

## SOMMARIO HARDWARE

Questo completa la nostra descrizione hardware dell'organizzazione interna del 6502. L'esatta struttura interna dei bus del 6502 non è importante a questo punto. Comunque il ruolo esatto di ogni registro è importante e dovrebbe essere pienamente compreso prima di proseguire la lettura. Quindi si prosegua solo se si ha familiarità con i concetti presentati, diversamente si suggerisce di rileggere ancora le parti essenziali di questo capitolo non appena queste sono utilizzate nei capitoli successivi. Si suggerisce di osservare ancora la Figura 2-2 e di assicurarsi la comprensione della funzione di ogni registro di questa illustrazione.

CAPITOLO 3

# TECNICHE DI PROGRAMMAZIONE DI BASE

## INTRODUZIONE

Lo scopo di questo capitolo è di presentare tutte le tecniche di base necessarie per scrivere un programma utilizzando il 6502. Questo capitolo introdurrà concetti addizionali come la gestione dei registri, i cicli e le subroutine. Esso sarà focalizzato sulle tecniche di programmazione utilizzando solo le risorse *interne* del 6502, cioè i registri. I programmi effettivi saranno sviluppati come programmi aritmetici. Questi programmi serviranno per illustrare i vari concetti presentati ed utilizzeranno istruzioni effettive. Si vedrà così come le istruzioni possono essere utilizzate per manipolare l'informazione tra la memoria la MPU, come pure per manipolare l'informazione all'interno della MPU stessa. Il capitolo successivo discuterà quindi i dettagli completi delle istruzioni disponibili sul 6502. Il Capitolo 6 presenterà le tecniche disponibili per manipolare l'informazione all'esterno del 6502: le tecniche d'ingresso/uscita.

In questo capitolo si apprenderà essenzialmente mediante esecuzione diretta. Esaminando programmi di complessità crescente si apprenderà il ruolo delle varie istruzioni, dei registri, e si applicheranno i concetti precedentemente sviluppati. Comunque un concetto importante che non verrà presentato è il concetto delle tecniche di indirizzamento. A causa della sua apparente complessità esso sarà presentato separatamente al Capitolo 5.

Si inizierà immediatamente scrivendo alcuni programmi per il 6502. Si partirà dai programmi aritmetici.

## PROGRAMMI ARITMETICI

I programmi aritmetici comprendono l'addizione, sottrazione, moltiplicazione e divisione. Il programma che verrà ora presentato opera su numeri interi. Questi interi possono essere binari positivi oppure anche

espressi nella notazione in complemento a 2 nel qual caso il bit più a sinistra è il bit del segno (Vedere il Capitolo 1 per la notazione in complemento a due).

## Addizione ad 8 bit

Si sommeranno due operandi ad 8 bit chiamati OP1 ed OP2, rispettivamente immagazzinati agli indirizzi di memoria ADR1 ed ADR2. La somma sarà chiamata RES e sarà immagazzinata all'indirizzo di memoria ADR3. Questo è illustrato in Figura 3-1. Il programma che eseguirà questa addizione è il seguente:

```
LDA     ADR1     CARICA OP1 IN A
ADC     ADR2     SOMMA OP2 AD OP1
STA     ADR3     CONSERVA IL RISULTATO AD ADR3
```

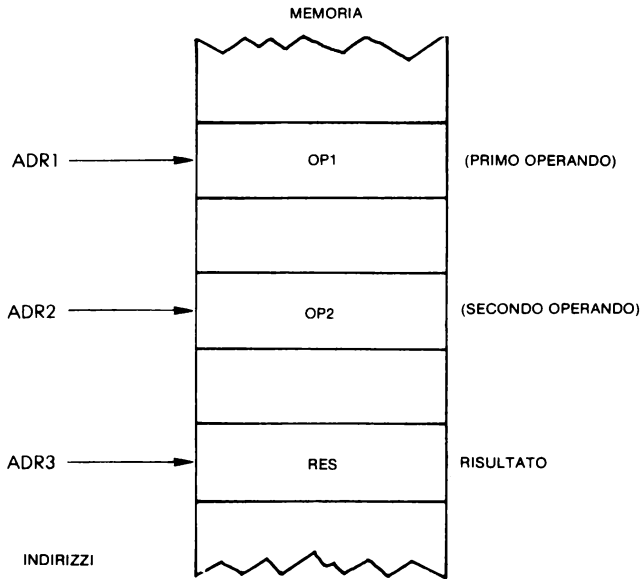



Figura 3.1: Addizione ad 8 bit: Res = OP1 + OP2

Questo è un programma di tre istruzioni. Ogni riga costituisce un'istruzione, in forma simbolica. Ogni istruzione sarà trasformata dal programma assemblatore in 1, 2 o 3 byte binari. Non si considererà

questa trasformazione qui, ma si osserverà solo la rappresentazione simbolica. In particolare la prima riga è un'istruzione LDA. LDA significa "carica l'accumulatore A dall'indirizzo che segue".

L'indirizzo specificato sulla prima riga è ADR1. ADR1 è una rappresentazione simbolica di un indirizzo effettivo a 16 bit. Da qualsiasi altra parte del programma sarà definito il simbolo ADR1. Esso potrebbe essere, per esempio, l'indirizzo 100.

L'istruzione LDA specifica "carica l'accumulatore A" (all'interno del 6502) dalla locazione di memoria 100. Questo si risolverà in un'operazione di lettura dall'indirizzo 100, i cui contenuti saranno trasmessi lungo il bus dati e depositati all'interno dell'accumulatore. Si ricorderà che le operazioni aritmetiche e logiche operano sull'accumulatore come uno degli operandi sorgente (per maggiori dettagli si faccia riferimento al Capitolo precedente). Poichè si desidera sommare assieme i due valori

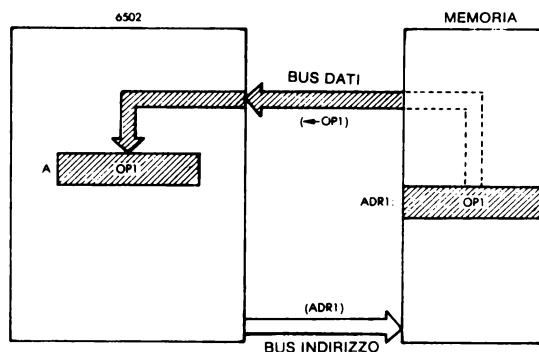



Figura 3.2: LDA ADR1: OP1 è caricato dalla memoria

OP1 ed OP2, innanzitutto si carica OP1 nell'accumulatore. Quindi si sarà in grado di sommare i contenuti dell'accumulatore (OP1) ad OP2.

Il campo più a destra di questa istruzione è detto *campo del commento.* Esso è ignorato dal processore ma viene fornito per la leggibilità del programma. Per comprendere cosa fa il programma è di importanza suprema impiegare dei buoni commenti.

Questa tecnica è la *documentazione* di un programma. Qui il commento è auto esplicativo: il valore di OP1, che è allocato all'indirizzo ADR1, viene caricato nell'accumulatore A.

Il risultato della prima istruzione è illustrato dalla Figura 3-2.

La seconda istruzione del programma in esame è:

ADC ADR2

Essa specifica "somma i contenuti della locazione di memoria ADR2 all'accumulatore". Con riferimento alla Figura 3-1, i contenuti della locazione di memoria ADR2 sono OP2, il secondo operando. I contenuti effettivi dell'accumulatore sono ora OP1, il primo operando. Come risultato dell'esecuzione della seconda istruzione, OP2 sarà prelevato dalla memoria e sommato ad OP1. La somma sarà depositata nell'accumulatore. Il lettore ricorderà che i risultati di un'operazione aritmetica, nel caso del 6502, sono ridepositi nell'accumulatore. Negli altri microprocessori, può essere possibile depositare questi risultati in altri registri o nella memoria.

La somma di OP1 ed OP2 è ora nell'accumulatore. Occorre ora trasferire i contenuti dell'accumulatore nella locazione di memoria ADR3 in modo da immagazzinare i risultati alla locazione richiesta. Anche qui il campo più a destra della seconda istruzione è semplicemente un campo commento che spiega il ruolo dell'istruzione (somma OP2 ad A).

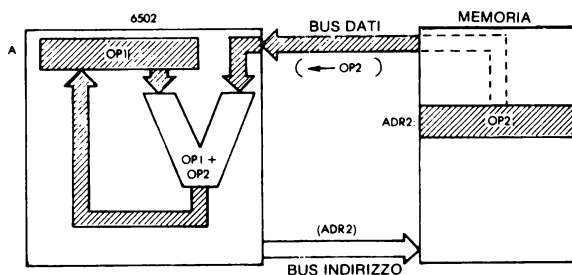



Figura 3.3: ADC ADR2

L'effetto della seconda istruzione è illustrato dalla Figura 3-3.

Si può verificare dalla Figura 3-3 che inizialmente l'accumulatore conteneva OP1. Dopo l'addizione un nuovo risultato è stato scritto nell'accumulatore. Questo è OP1 + OP2. I contenuti di qualsiasi registro all'interno del sistema, come pure di qualsiasi locazione di memoria, rimangono invariati quando viene eseguita un'operazione di lettura. In altre parole, *la lettura dei contenuti di un registro o di una locazione di memoria non cambia i suoi contenuti.* Soltanto, ed esclusivamente, un'o-

perazione di scrittura cambierà i contenuti di un registro. In questo esempio i contenuti delle locazioni di memoria ADR1 ed ADR2 sono invariati. Comunque dopo la seconda istruzione di questo programma, i contenuti dell'accumulatore sono stati modificati poichè l'uscita della ALU è stata scritta nell'accumulatore. I suoi contenuti precedenti sono andati persi.

Si conserverà ora questo risultato all'indirizzo ADR3 e questa semplice addizione sarà così completata.

La terza istruzione specifica: STA ADR3. Questo significa "immagazzina i contenuti dell'accumulatore A all'indirizzo ADR3". Questo è auto-esplicativo ed è illustrato in Figura 3-4.

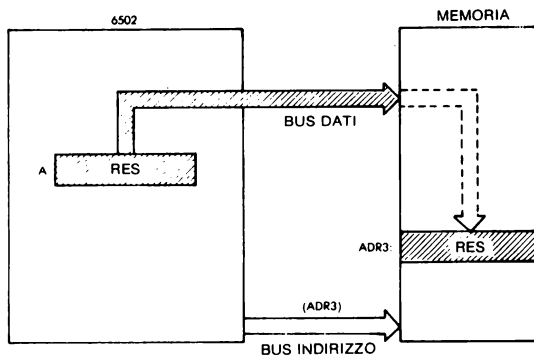



Figura 3.4: STA ADR3 (immagazzina in memoria i contenuti dell'accumulatore)

## *Peculiarità del 6502*

Il precedente programma di tre istruzioni sarebbe davvero un programma completo per la maggior parte dei microprocessori. Comunque esistono due peculiarità del 6502 che normalmente richiederanno due istruzioni addizionali.

Primo, l'istruzione ADC in realtà significa "somma *con* carry" piuttosto che "somma". La differenza sta nel fatto che una normale addizione somma due numeri assieme. Un'addizione-con-carry somma due numeri assieme più il valore del bit carry. Poichè qui si stanno sommando due numeri di 8 bit il carry non dovrebbe essere utilizzato. Ora all'inizio dell'addizione non si conosce necessariamente la condizione del bit carry (esso può essere stato posto ad uno da un'istruzione precedente), si deve quindi azzerarlo. Questo sarà eseguito dall'istruzione CLC "azzera carry".

Sfortunatamente il 6502 non possiede entrambi i tipi di operazione di addizione. Esso possiede solo l'operazione ADC. Come risultato, per singole addizioni ad 8 bit, una precauzione necessaria è quella di azzerare sempre il bit carry. Questo non è uno svantaggio significativo ma non deve essere dimenticato.

La seconda peculiarità del 6502 concerne il fatto che esso è equipaggiato con istruzioni decimali potenti che saranno impiegate al paragrafo successivo con l'aritmetica BCD. Il 6502 funziona sempre in uno dei due modi: binario o decimale. Lo stato in cui si trova è condizionato dal bit di stato, il bit "D" (del registro P). Poichè in questo esempio si sta considerando un funzionamento un modo binario è necessario assicurarsi che D sia posizionato correttamente. Questo sarà fatto da un'istruzione CLD, che azzererà il bit D. Naturalmente se tutte le operazioni aritmetiche all'interno del sistema sono eseguite in binario il bit D sarà azzerato una sola volta e per tutte all'inizio del programma e non sarà necessario posizionarlo tutte le volte. Perciò questa istruzione può di fatto essere omessa nella maggior parte dei programmi. Comunque il lettore che farà pratica di questi esercizi su un calcolatore, può passare da esercizi in BCD ad esercizi in binario e questa ulteriore istruzione inclusa qui deve apparire almeno una volta prima dell'esecuzione di qualsiasi addizione binaria.

Per riassumere: il programma ad 8 bit completo e sicuro è ora:

```
CLC                     AZZERA IL BIT CARRY
CLD                     AZZERA IL BIT DECIMALE
LDA        ADR1         CARICA OP1 IN A
ADC        ADR2         SOMMA OP2 AD OP1
STA        ADR3         CONSERVA RES AD ADR3
```

Si possono utilizzare indirizzi fisici effettivi invece di ADR1, ADR2 ed ADR3. Se si desidera mantenere gli indirizzi simbolici sarà necessario utilizzare le cosiddette "pseudo-operazioni" che specificano il valore di questi indirizzi simbolici cosicchè il programma assembly può, durante la traduzione, sostituire gli indirizzi fisici effettivi. Tali pseudo-operazioni sarebbero per esempio:

```
ADR1 = $ 100
ADR2 = $ 120
ADR3 = $ 200
```

**Esercizio 3.1:** *Facendo riferimento soltanto alla lista delle istruzioni alla fine del libro, si scriva un programma che sommi due numeri immagazzinati alle locazioni di memoria LOC1 e LOC2. Si depositino i risultati alla*

*locazione di memoria LOC3. Quindi si confronti il programma con quello precedente.*

## Addizione a 16 bit

Un'addizione ad 8 bit consentirà solo l'addizione di numeri ad 8 bit, cioè numeri tra 0 e 225, se è utilizzato il binario assoluto. Per applicazioni più pratiche è necessario utilizzare una *precisione multipla* per sommare numeri maggiori o uguali a 16 bit. Si presenteranno qui degli esempi di aritmetica a 16 bit. Essi possono essere facilmente estesi a 24, 32 oppure più bit. (Ma sempre multipli di 8 bit). Si assumerà che il primo operando sia immagazzinato alla locazione di memoria ADR1 ed ADR1-1. Poichè questa volta OP1 è un numero a 16 bit, esso richiederà due locazioni di memoria ad 8 bit.

Analogamente OP2 sarà depositato agli indirizzi di memoria ADR2 ed ADR2-1. Il risultato deve essere depositato agli indirizzi di memoria ADR3 ed ADR3-1. Questo è illustrato in Figura 3-5.

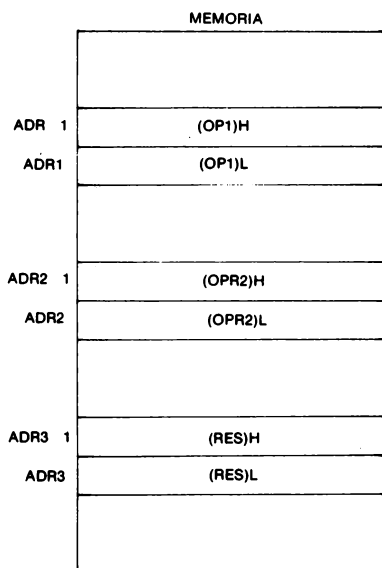



Figura 3.5: Addizione a 16 bit: gli operandi

La logica di questo programma è esattamente la stessa di quello precedente. Prima sarà sommata la metà di basso ordine degli operandi

poichè il microprocessore può sommare soltanto su 8 bit alla volta. Qualsiasi riporto generato dall'addizione di questi due byte di basso ordine sarà automaticamente immagazzinato nel bit carry interno ("C"). Quindi le metà di ordine elevato dei due operandi saranno sommate insieme con qualsiasi carry ed il risultato sarà conservato nella memoria.

Il programma è il seguente:

```
CLC
CLD
LDA       ADR1       META' BASSA DI OP1
ADC       ADR2       (OP + OP2) BASSO
STA       ADR3       CONSERVA LA META' BASSA DI RES
LDA       ADR1-1     META' ALTA DI OP1
ADC       ADR2-1     (OP1 + OP2) ALTO + RIPORTO
STA       ADR3-1     CONSERVA LA META' ALTA DI RES
```

Le prime due istruzioni di questo programma sono utilizzate per sicurezza: CLC, CLD. Il loro ruolo è stato spiegato nel paragrafo precedente. Si esamini ora il programma: le successive tre istruzioni sono essenzialmente identiche a quelle dell'addizione della metà meno significativa (bit da 0 a 7) di OP1 ed OP2. La somma, chiamata RES, è immagazzinata alla locazione di memoria ADR3.

Automaticamente, ogni volta che viene eseguita un'addizione, qualsiasi riporto risultante è conservato nel bit carry del registro dei flag (registro P). Se i due numeri non generano nessun riporto, il valore del bit carry sarà zero. Se i due numeri generano un riporto allora il bit C sarà uguale ad 1.

Le successive tre istruzioni del programma sono inoltre essenzialmente identiche alla precedente addizione ad 8 bit. Esse sommano assieme le metà più significative (i bit da 8 a 15) di OP1 ed OP2, più qualsiasi carry, ed immagazzinano il risultato all'indirizzo ADR3-1. Dopo che è stato eseguito questo programma, il risultato a 16 bit è immagazzinato alle locazioni di memoria ADR3 ed ADR3-1.

Qui è stato assunto che nessun riporto sia generato da questa addizione a 16 bit. Si è assunto infatti che il risultato sia un numero a 16 bit. Se il programmatore sospetta per qualunque ragione che il risultato possa avere 17 bit allora dovrebbe essere inserita un'istruzione addizionale per verificare il bit carry dopo questa addizione. La locazione degli operandi nella memoria è illustrata in Figura 3-5.

Nota: qui è stato assunto che la parte più alta dell'operando sia immagazzinata "alla sommità" della parte più bassa cioè all'indirizzo di memoria più basso. Questo può non essere generale. Infatti gli indirizzi

sono immagazzinati in modo opposto: la parte bassa è memorizzata per prima nella memoria e poi la parte più alta è immagazzinata nella successiva locazione di memoria. Per utilizzare una convenzione comune per i dati e gli indirizzi si raccomanda che anche i dati siano conservati con la parte più bassa sopra la parte più alta. Questo è illustrato in Figura 3-6 a e b.

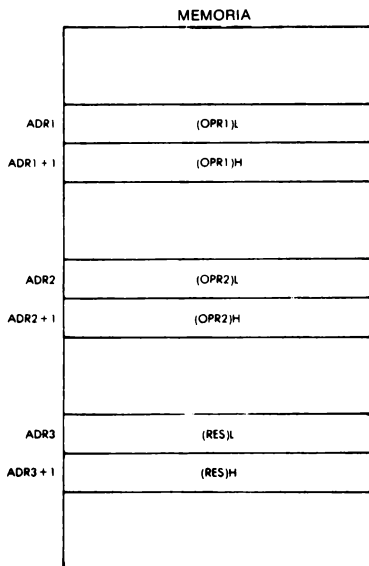



Figura 3.6a: Immagazzinamento degli operandi di ordine inverso

**Esercizio 3.2:** *Si riscriva il programma dell'addizione a 16 bit con lo schema di memoria indicato in Figura 3-6a.*

**Esercizio 3.3:** *Si assuma ora che ADR1 non punti alla metà più bassa di OPR1 (vedere Figura 3-6a), ma punti alla parte più alta di OPR1. Questo è illustrato in Figura 3-6b. Si scriva nuovamente il programma corrispondente.*

Il programmatore deve decidere come immagazzinare i numeri a 16 bit (cioè prima la parte bassa o quella alta) ed anche se l'indirizzo di riferimento punti alla metà più bassa o più alta di tali numeri. Questa è la prima di molte scelte che si imparerà ad eseguire quando si progettano algoritmi oppure strutture dati.

Figura 3.6b: Puntamento al byte elevato

Si è così imparato ad eseguire l'addizione binaria. Si considererà ora la sottrazione.

## Sottrazione di numeri a 16 Bit

L'esecuzione di una sottrazione ad 8 bit è troppo semplice. Si eseguirà ora per esercizio una sottrazione a 16 bit. Come al solito i numeri che si considerano, OPR1 ed OPR2, sono immagazzinati agli indirizzi ADR1 ed ADR2. Lo schema di memoria sarà assunto essere quello di Figura 3-6. Per sottrarre si eseguirà l'operazione di sottrazione (SBC) invece di un'operazione di addizione (ADC). L'unica variazione, rispetto all'addizione, è che si utilizzerà un'istruzione SEC all'inizio del programma invece di un CLC. SEC significa "pone ad 1 carry". Questo indica una condizione di "non prestito". Il resto del programma è identico a quello dell'addizione. Il programma è il seguente:

```
CLD
SEC                     PONE CARRY AD 1
LDA        ADR1         (OPR1) L IN A
SBC        ADR2         (OPR1) L - (OPR2) L
```

```
STA     ADR3        IMMAGAZZINA (RESULT) L
LDA     ADR1 + 1    (OPR1) H IN A
SBC     ADR2 + 1    (OPR1) H - (OPR2) H
STA     ADR3 + 1    IMMAGAZZINA (RESULT) H
```

**Esercizio 3.4:** *Si scriva il programma della sottrazione per gli operandi ad 8 bit.*

Si deve ricordare che nel caso dell'aritmetica in complemento a 2 il valore finale del flag carry non è significativo. Se si è verificata una condizione di overflow come risultato della sottrazione allora viene posto ad uno il bit di overflow (bit V) del registro dei flag. Questo può quindi essere verificato.

Gli esempi appena presentati sono semplici addizioni binarie. Comunque può essere necessaria un altro tipo di addizione: è l'addizione BCD.

## ARITMETICA BCD

### Addizione BCD ad 8 Bit

Il concetto dell'aritmetica BCD è stato presentato al Capitolo 1. Essa è utilizzata essenzialmente per applicazioni commerciali dove è imperativo conservare ogni digit significativo di un risultato. Nella notazione BCD, viene utilizzato un nibble di 4 bit per immagazzinare un digit decimale (da 0 a 9). Ne risulta che ogni byte di 8 bit può immagazzinare due digit BCD. (Questo è chiamato *BCD packed*). Si sommino ora due byte contenenti due digit BCD ciascuno.

Per definire i problemi si provino innanzi tutto alcuni esempi numerici.

Si sommi "01" e "02":

"01" è rappresentato da 0000 0001.

"02" è rappresentto da 0000 0010.

Il risultato è 0000 0011.

Questa è la rappresentazione BCD di "03". (Per assicurarsi dell'equivalente BCD si consulti la tabella di conversione alla fine del libro). Le cose sono molto semplici in questo caso.
Si consideri un altro esempio.

"08" è rappresentato da 0000 1000.
"03" è rappresentato da 0000 0011.

**Esercizio 3.5:** *Calcolare la somma dei due numeri precedenti nella rappresentazione BCD. Che cosa si ottiene? (la risposta è riportata di seguito).*

Se è stato ottenuto 0000 1011 è stata calcolata la somma *binaria* di "8" e "3". Si è infatti ottenuto "11" in *binario.* Sfortunatamente "1011" è un *codice BCD non consentito.* Si doveva ottenere la rappresentazione BCD di "11" cioè "0001 0001"!

Il problema deriva dal fatto che la rappresentazione BCD utilizza solo le prime dieci combinazioni di 4 digit in modo da codificare i simboli decimali da "0" a "9". Le rimanenti sei combinazioni di 4 digit sono inutilizzate ed il valore non consentito "1011" è una di queste combinazioni. In altre parole ogni volta che la somma di due digit binari è maggiore di "9" si deve aggiungere "6" al risultato in modo da saltare i 6 codici inutilizzati. Si sommi quindi la rappresentazione binaria di "6" ad "1011":

```
                      1011    (risultato binario non consentito)
                    + 0110    (+ 6)
                 ---------
Il risultato è:  0001 0001.
```

Questo è, infatti, "11" nella notazione BCD! Quindi è stato ottenuto il risultato corretto.

Questo esempio illustra una delle difficoltà di base dell'impiego del BCD. Occorre compensare 6 codici inutilizzati. Nella maggior parte dei microprocessori, un'istruzione speciale, chiamata "aggiustamento decimale" deve essere utilizzata per aggiustare il risultato di un'addizione binaria. (Somma 6 se il risultato è maggiore di 9). Nel caso del 6502 l'istruzione ADC fa questo automaticamente. Questo è un indubbio vantaggio del 6502 quando opera nell'aritmetica BCD.

Il problema successivo è illustrato dallo stesso esempio. Nell'esempio considerato il riporto sarà generato dal digit BCD più basso (quello più a destra) in quello più a sinistra. Questo riporto interno deve essere considerato e sommato al secondo digit BCD. L'istruzione di addizione del 6502 fa questo automaticamente. Comunque è spesso conveniente rivelare questo riporto interno dal bit 3 al bit 4 (il "riporto intermedio"). Non esistono flag per questo scopo nel 6502.

Infine, come nel caso dell'addizione binaria, occorre utilizzare le usuali istruzioni SED e CLC prima dell'esecuzione dell'addizione BCD vera e propria. Come esempio si riporta un programma per la somma

BCD dei numeri "11" e "22":

```
CLC                         AZZERA CARRY
SED                         POSIZIONA MODO DECIMALE
LDA          # $ 11         BCD LETTERALE "11"
ADC          # $ 22         BCD LETTERALE "22"
STA          ADR
```

In questo programma sono stati utilizzati due nuovi simboli: "#" ed "$". Il simbolo "#" denota che segue un "letterale" (o costante). Il segno "$", all'interno del campo operando dell'istruzione, specifica che i dati che seguono sono espressi in notazione esadecimale. Le notazioni esadecimali e BCD per i digit da "0" a "9" sono identiche. Qui si desidera sommare i letterali (o costanti) "11" e "22". Il risultato è immagazzinato all'indirizzo ADR. Quando l'operando è specificato come parte dell'istruzione, come nell'esempio precedente, si ha il cosiddetto *indirizzamento immediato.* (I vari modi di indirizzamento saranno discussi in dettaglio al Capitolo 5).

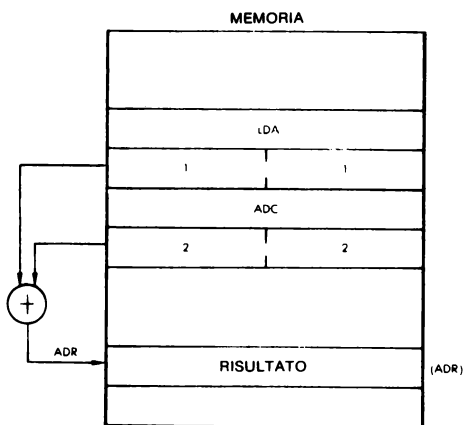



Figura 3.7: Immagazzinamento dei digit BCD

L'immagazzinamento del risultato ad un indirizzo specificato, come con STA ADR, è chiamato *indirizzamento assoluto* quando ADR rappresenta un regolare indirizzo a 16 bit.

**Esercizio 3.6:** ***È possibile spostare l'istruzione CLC nel programma sotto l'istruzione LDA?***

### Sottrazione BCD

La sottrazione BCD è apparentemente complessa. Per eseguire una sottrazione BCD si deve sommare il complemento a 10 del numero proprio come occorre sommare il complemento a 2 di un numero per eseguire la sottrazione binaria. Il complemento a 10 si ottiene calcolando il complemento a 9 ed aggiungendo 1. Questo richiede tipicamente tre o quattro operazioni su un microprocessore convenzionale. Comunque il 6502 è equipaggiato con una speciale istruzione di sottrazione BCD che esegue questo con una semplice istruzione! Naturalmente, e come nell'esempio binario, il programma sarà preceduto dalle istruzioni SED (che seleziona il modo decimale, se non è stato fatto precedentemente) ed SEC che pone il carry ad 1. Così il programma per sottrare "25" a "26" in BCD è il seguente:

```
SED                 PONE MODO DECIMALE
SEC                 PONE CARRY
LDA     # $26       CARICA IL BCD 26
SBC     # $25       MENO IL BCD 25
STA     ADR         IMMAGAZZINA IL RISULTATO
```

### Addizione BCD a 16 bit

L'addizione a 16 bit viene eseguita allo stesso modo del caso binario. Il programma per tale addizione è il seguente:

```
CLC
SED
LDA     ADR1
ADC     ADR2
STA     ADR3
LDA     ADR1-1
ADC     ADR2-1
STA     ADR3-1
```

**Esercizio 3.7:** *Si confronti il programma precedente con quello utilizzato per l'addizione binaria a 16 bit. Qual'è la differenza?*

**Esercizio 3.8:** *Si scriva il programma per la sottrazione per il BCD a 16 bit. (Non si utilizzi CLC ed ADC!).*

### Flag BCD

Nel modo BCD il flag carry durante un'addizione indica che il risultato è maggiore di 99. Questo non è come nella situazione del complemento a

2 poichè i digit BCD sono rappresentati in binario vero. Inversamente l'assenza del flag carry durante una sottrazione indica un prestito.

**Suggerimenti di Programmazione per la Somma e la Sottrazione**

—Azzerare sempre il flag carry prima di eseguire un'addizione.
—Porre sempre ad 1 il flag carry prima di eseguire una sottrazione.
—Porre il modo appropriato: binario o decimale.

*Tipi di Istruzioni*

Sono stati utilizzati tre tipi di istruzioni del microprocessore. Sono state impiegate LDA e STA che, rispettivamente, caricano l'accumulatore da un indirizzo di memoria, ed immagazzinano i suoi contenuti all'indirizzo specificato. Queste sono due istruzioni di *trasferimento dati.*

Successivamente sono state utilizzate istruzioni *aritmetiche,* come ADC ed SBC. Queste eseguono rispettivamente le operazioni di addizione e sottrazione. Ulteriori istruzioni ALU saranno introdotte nel corso di questo capitolo.

Infine sono state utilizzate istruzioni, come CLC e SED, ed altre, che manipolano i bit di flag (rispettivamente il bit carry ed il bit decimale negli esempi considerati). Queste sono istruzioni di *manipolazione di stato* o di controllo. Un'estesa descrizione delle istruzioni del 6502 sarà presentata al Capitolo 4.

Ancora altri tipi di istruzioni sono disponibili all'interno del microprocessore e non sono ancora state utilizzate.

Queste sono in particolare le istruzioni di "diramazione" e di "salto" che modificheranno l'ordine secondo il quale il programma deve essere eseguito. Questo nuovo tipo di istruzioni sarà introdotto nell'esempio successivo.

**Moltiplicazione**

Si consideri ora un problema aritmetico più complesso: la moltiplicazione di numeri binari. Per introdurre l'algoritmo di una moltiplicazione binaria si inizia esaminando l'ordinaria moltiplicazione decimale: Si moltiplicherà 12 × 23.

| | |
|---:|---|
| 12 | (Moltiplicando) |
| × 23 | (Moltiplicatore) |
| 36 | (Prodotto parziale) |
| + 24 | |
| = 276 | Risultato finale |

La moltiplicazione è eseguita moltiplicando la cifra più a destra del moltiplicatore, col moltiplicando, cioè "3" × 12". Il prodotto parziale è "36". Quindi si moltiplica la cifra successiva del moltiplicatore cioè "2" per "12". "24" è sommato al prodotto parziale.

Ma c'è un'ulteriore operazione: 24 è *spostato a sinistra* di una posizione. In modo equivalente si potrebbe dire che il prodotto parziale (36) è stato *spostato a destra di una posizione* prima di sommarlo.

I due numeri, correttamente spostati, sono poi sommati e la somma è 276. Questo è semplice. Si consideri ora la moltiplicazione binaria. La moltiplicazione binaria è eseguita esattamente nello stesso modo.

Si consideri un esempio. Si moltiplicherà 5 × 3:

| | | |
|---|---:|---|
| (5) | 101 | (MPD) |
| (3) | × 011 | (MPR) |
| | 101 | (PP) |
| | 101 | |
| | 000 | |
| (15) | 01111 | (RES) |

Per eseguire la moltiplicazione si opera esattamente come sopra. La

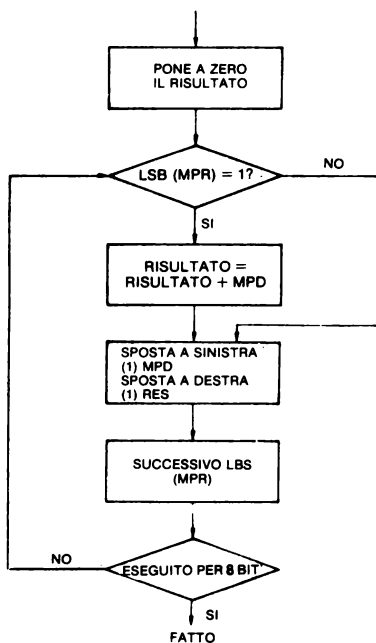



Figura 3.8: L'algoritmo di base della moltiplicazione: diagrammi-di flusso

rappresentazione formale di questo algoritmo appare in Figua 3-8. Questo è un diagramma di flusso per l'algoritmo, il primo diagramma di flusso. Esaminaniamolo più in dettaglio.

Questo diagramma di flusso è una rappresentazione simbolica dell'algoritmo appena considerato. Ogni rettangolo rappresenta un ordine da eseguire. Esso sarà tradotto in una o più istruzioni di programma. Ogni simbolo a forma di rombo rappresenta un test da eseguire. Questo sarà un punto di diramazione del programma. Se il test si verifica si entra in un'altra locazione. Il concetto di diramazione sarà spiegato successivamente nel programma stesso. Il lettore dovrebbe ora esaminare questo diagramma di flusso ed accertare che esso rappresenta esattamente l'algoritmo presentato precedentemente. Si noti che c'è una freccia che esce dall'ultimo rombo in fondo al diagramma di flusso ed una che entra nel primo rombo in alto. Questo perchè la stessa porzione di diagramma di flusso sarà eseguita otto volte, una volta per ogni bit del moltiplicatore. Una situazione di questo genere dove l'esecuzione riparte dallo stesso punto è detta un *ciclo di programma (loop)* per ovvie ragioni.

**Esercizio 3.9:** *Si moltiplichi in binario "4" per "7" utilizzando il diagramma di flusso e verificando che si ottiene "28". Se ciò non accade si provi ancora. Solo se è stato ottenuto il risultato corretto si è in grado di tradurre questo diagramma di flusso in un programma.*

Si traduca ora questo diagramma di flusso in un programma per il 6502. Il programma completo appare in figura 3-9. Si studierà questo in dettaglio. Come si ricorderà dal Capitolo 1, la programmazione in questo caso consiste nella traduzione del diagramma di flusso di Figura 3-8 nel programma di Figura 3-9. Ciascun blocco del diagramma di flusso sarà tradotto in una o più istruzioni.

È stato assunto che MPR ed MPD abbiano già un valore.

Il primo blocco del diagramma di flusso è un blocco di *inizializzazione.* È necessario porre a "0" un certo numero di registri o locazioni di memoria poichè questo programma li utilizzerà. I registri che saranno utilizzati dal programma della moltiplicazione appaiono in Figura 3-10.

Sulla sinistra dell'istruzione appare la porzione rilevante del microprocessore 6502. Sulla destra dell'illustrazione appare una sezione rilevante dalla memoria. Qui si assumerà che gli indirizzi di memoria aumentino dall'alto al basso dell'illustrazione. Naturalmente si potrebbe utilizzare la convenzione opposta. Il registro X, riportato all'estrema sinistra (uno dei due registri indice del 6502) sarà utilizzato come *conta-*

```
        LDA   # 0         AZZERA L'ACCUMULATORE
        STA   TMP         AZZERA QUESTO INDIRIZZO
        STA   RESAD       AZZERA
        STA   RESAD + 1   AZZERA
        LDX   # 8         X È IL CONTATORE
MULT    LSR   MPRAD       SPOSTA MPR A DESTRA
        BCC   NO ADD      TEST DEL BIT DEL CARRY
        LDA   RESAD       CARICA CON RES BASSO
        CLC               PREPARA A SOMMARE
        ADC   MPDAD       SOMMA MPD A RES
        STA   RESAD       CONSERVA IL RISULTATO
        LDA   RESAD + 1   SOMMA IL RESTO DI MPD SPOSTATO
        ADC   TMP
        STA   RESAD + 1
NOADD   ASL   MPDAD       SPOSTA MPD A SINISTRA
        ROL   TMP         CONSERVA IL BIT DA MPD
        DEX               DECREMENTA IL CONTATORE
        BNE   MULT        RIPETI SE CONTATORE # 0
```

Figura 3.9: Moltiplicazione 8x8 bit

*tore.* Poichè si sta eseguendo una moltiplicazione ad 8 bit si devono verificare gli 8 bit del moltiplicatore. Sfortunatamente non ci sono istruzioni nel 6502 che consentono di provare detti bit in sequenza. I soli bit che possono essere verificati convenientemente sono i flag del registro di stato. Come risultato di questa limitazione della maggior parte dei microprocessori, per verificare successivamente tutti i bit del moltiplicatore sarà necessario trasferire il valore del moltiplicatore nell'accumulatore. Quindi i contenuti dell'accumulatore saranno fatti scorrere a destra.

Un'istruzione di scorrimento muove ogni bit del registro di una posizione a destra oppure a sinistra. L'effetto di un'operazione di scorrimento è illustrato in Figura 3-10. Esistono molte varianti possibili in dipendenza del bit che entra nel registro ma queste differenze saranno discusse al Capitolo 4 (set di istruzioni del 6502).

Si ritorni alle successive verifiche di ciascuno degli 8 bit del moltiplicatore. Poichè si può verificare facilmente il bit carry, il moltiplicatore sarà spostato di una posizione 8 volte. Ogni volta il suo bit più a destra cadrà nel bit carry e sarà verificato. Il problema successivo da risolvere è che il prodotto parziale che sarà accumulato durante le addizioni successive richiederà 16 bit. La moltiplicazione di due numeri ad 8 bit può produrre un risultato a 16 bit. Questo perché $2^8 \times 2^8 = 2^{16}$. È quindi necessario

Figura 3.10: Moltiplicazione i registri

riservare 16 bit per questo risultato. Sfortunatamente il 6502 ha veramente pochi registri interni cosicchè questo prodotto parziale non può essere memorizzato all'interno del 6502 stesso. Infatti, a causa del limitato numero di registri non si è in grado di immagazzinare il moltiplicatore, il moltiplicando oppure i prodotti parziali all'interno del 6502. Essi saranno immagazzinati in memoria. Questo originerà un'esecuzione più lenta di quella che sarebbe possibile ottenere memorizzandoli tutti nei registri interni. Questa è una limitazione del 6502. L'area di memoria utilizzata per la moltiplicazione appare sulla parte destra della Figura 3-10. In alto si può vedere la parola di memoria allocata per il moltiplica-

Figura 3.11: Scorrimento e rotazione

tore. Si assumerà, per esempio, che esso contenga "3" in binario. L'indirizzo di questa locazione di memoria è MPRAD. Sotto a questo si trova un "temporaneo" il cui indirizzo è TMP. Il ruolo di questa locazione sarà chiarito di seguito. Si sposterà il moltiplicando a sinistra nella locazione principale aggiungendolo al prodotto parziale. Il moltiplicando è successivo e si assumerà contenere il valore "5" in binario. Il suo indirizzo è MPDAD.

Infine, in fondo alla memoria, si trovano due parole allocate per il prodotto parziale ovvero il risultato. Il loro indirizzo è RESAD.

Queste locazioni di memoria saranno i "registri di lavoro" e la parola "registro" può essere utilizzata come sinonimo di "locazione" in questo contesto.

La freccia che compare nella parte destra in alto dell'illustrazione che fa entrare MPR nel bit C è un modo simbolico per mostrare come il moltiplicatore sia fatto scorrere nel bit carry. Naturalmente questo bit carry è fisicamente contenuto all'interno del 6502 e non all'interno della memoria.

Si ritorni ora al programma di Figura 3-9. Le prime cinque sono istruzioni di inizializzazione.

Le prime quattro istruzioni azzereranno i contenuti dei "registri" TMP, RESAD e RESAD + 1. Si verifichi questo.

LDA # 0

Questa istruzione carica l'accumulatore con il valore letterale "0". Come risultato di questa istruzione l'accumulatore conterrà "00000000".

I contenuti dell'accumulatore verranno ora utilizzati per azzerare i tre "registri" della memoria. Occorre ricordare che la lettura di un valore da un registro non altera il suo contenuto. Così è possibile leggere il contenuto di un registro tante volte quanto è necessario. I suoi contenuti non vengono cambiati dall'operazione di lettura. Si proceda:

STA TMP

Questa istruzione immagazzina i contenuti dell'accumulatore nella locazione di memoria TMP. Si faccia riferimento alla Figura 3-11 per capire il flusso dei dati nel sistema. L'accumulatore contiene "00000000". Il risultato di questa istruzione sarà la scrittura di tutti zeri nella locazione di memoria TMP. Si ricordi che i contenuti dell'accumulatore rimangono 0 dopo un'operazione di lettura. Sono invariati. Si sta per utilizzarli ancora.

STA RESAD

Questa istruzione opera esattamente come la precedente ed azzera i contenuti dell'indirizzo RESAD. Analogamente opera:

STA RESAD + 1

Infine si azzera la locazione di memoria RESAD + 1 che è stata riservata per memorizzare la parte alta del risultato. (La parte alta sono i bit 8-15; la parte bassa sono i bit 0-7).

Infine, per arrestare lo scorrimento dei bit del moltiplicatore all'istante corretto, è necessario contare il numero di scorrimenti che sono stati eseguiti. Sono necessari otto scorrimenti. Il registro X sarà utilizzato come contatore ed inizializzato al valore "8". Ogni volta che viene eseguito uno scorrimento, i contenuti di questo contatore saranno decrementati di 1. Quando il valore del contatore diventa "0" la moltiplicazione è terminata. Si inizializza questo registro ad "8":

LDX # 8

Questa istruzione carica il letterale "8" nel registro X.

Con riferimento al diagramma di flusso di Figura 3-8 si deve verificare il bit meno significativo del moltiplicatore. È stato indicato precedentemente che questa prova non può essere eseguita in una singola istruzione. Occorre utilizzare due istruzioni. Prima il moltiplicatore sarà fatto scorrere a destra, poi il bit che esce sarà verificato. Questo è il bit carry. Si esegua quest'operazione:

LSR MPRAD

Questa istruzione è uno *Spostamento Logico a Destra* dei contenuti della locazione di memoria MPRAD.

**Esercizio 3.10:** *Assumendo che il moltiplicatore nell'esempio sia "3" qual'è il bit che cade dalla parte della locazione di memoria MPRAD? (In altre parole quale sarà il valore del carry dopo questo scorrimento?).*

La successiva istruzione verifica il valore del bit carry:

BCC NOADD

Questa istruzione significa "Se il Carry è Zero "vai" all'indirizzo NOADD.

Questa è la prima volta che si incontra un'istruzione di diramazione. Tutti i programmi fin'ora considerati erano strettamente sequenziali. Ogni istruzione era eseguita dopo la precedente in ordine sequenziale. Per capire l'utilizzazione di test logici, come quello del bit carry, si deve

essere in grado di eseguire istruzioni dovunque nel programma dopo il test. L'istruzione di diramazione esegue appunto tale funzione. Si esaminerà il valore del bit carry. Se il carry era "0", cioè azzerato, allora l'esecuzione del progamma proseguirà all'indirizzo NOADD. Questo significa che la successiva istruzione eseguita dopo BCC sarà l'istruzione all'indirizzo NOADD, se il test è soddisfatto.

Altrimenti, se il test non è soddisfatto, non si verificherà alcuna diramazione e sarà eseguita l'istruzione sequenziale successiva BCC NOADD.

NOADD necessita di un'ulteriore spiegazione: questa è una *label simbolica.* Essa rappresenta fisicamente un indirizzo effettivo all'interno della memoria. Per convenienza del programmatore, il programma assemblatore consente l'utilizzazione di nomi simbolici al posto di indirizzi effettivi. Durante il processo assembly, l'assemblatore sostituirà l'indirizzo fisico reale al posto del simbolo "NOADD". Questo migliora la leggibilità del programma in modo sostanziale e consente anche al programmatore di inserire istruzioni addizionali tra il punto di diramazione e NOADD, senza dover riscrivere ogni cosa. Questi artifici saranno studiati in maggior dettaglio al Capitolo 9 sull'assemblatore.

Se il test non è soddisfatto viene eseguita l'istruzione sequenzialmente successiva nel programma. Si esamineranno ora entrambe le alternative.

*Alternativa 1: il carry sia "1".*

Se il carry è 1 il test da BCD non è soddisfatto e viene eseguita l'istruzione immediatamente sucessiva BCC in ordine sequenziale.

LDA RESAD

*Alternativa 2: il carry sia "0".*

Il testo è soddisfatto e la successiva istruzione è quella con la label "NOADD".

Con riferimento alla Figura 3-8, il diagramma di flusso specifica che, se il bit carry era 1, il moltiplicando deve essere sommato al prodotto parziale (nel caso considerato i registri RES). Inoltre occorre eseguire uno scorrimento. Il prodotto parziale deve essere mosso di una posizione a destra ovvero il moltiplicando deve essere mosso di una posizione a sinistra. Si adotterà qui la convenzione normalmente impiegata nell'esecuzione manuale della moltiplicazione e si muoverà il moltiplicando di una posizione a sinistra.

Il moltiplicando è contenuto nei registri TMP ed MPDAD. (Per semplicità saranno normalmente chiamate "registri" le locazioni di memoria). I 16 bit del prodotto parziale sono contenuti agli indirizzi di memoria RESAD e RESAD + 1.

Per spiegare questo si assuma che il moltiplicando sia "5". I vari registri appaiono in Figura 3-10.

Si devono semplicemente addizionare due numeri a 16 bit. Questo è un problema che si è già imparato a risolvere. (Se si ha qualche dubbio si faccia riferimento al precedente paragrafo sull'addizione a 16 bit). Si sommeranno prima i byte di basso ordine e poi quelli di ordine elevato. Si procede così:

LDA RESAD

L'accumulatore è caricato con la parte bassa di RES.

CLC

Prima di qualsiasi addizione il 6502 richiede che il bit cary sia azzerato. Qui è particolarmente importante perchè si sa che il bit carry era stato posto ad 1. Perciò esso deve essere azzerato.

ADC MPDAD

Il moltiplicando è sommato all'accumulatore, che contiene (RES) BASSO.

STA RESAD

Il risultato dell'addizione è conservato all'appropriata locazione di memoria (RES) BASSO. Viene poi eseguita la seconda metà dell'addizione. Durante l'esecuzione del controllo manuale di questo programma non si dimentichi che l'addizione porrà il bit carry. Il carry sarà posto a "0" o ad "1" in dipendenza del risultato dell'addizione. Qualsiasi riporto che deve essere generato sarà portato automaticamente nella parte di ordine elevato del risultato.

Si completa ora l'addizione:

```
LDA    RESAD + 1
ADC    TMP
STA    RESAD + 1
```

Queste istruzioni completano l'addizione a 16 bit. Si è ora sommato il moltiplicando a RES. Occorre ora spostarlo di una posizione a sinistra prima dell'addizione successiva. Si può anche considerare lo spostamento del moltiplicando di una posizione a sinistra *prima* dell'addizione, eccetto che per la prima volta. Questa è una delle molte scelte di programmazione sempre aperte al programmatore.

Si faccia scorrere il moltiplicando a sinistra:

NOADD ASL MPDAD

Questa istruzione è uno "Spostamento Aritmetico a Sinistra".

Essa sposterà di una posizione a sinistra i contenuti della locazione di memoria MPDAD che contiene la parte bassa del moltiplicando. Questo non basta. Non ci si può permettere di perdere il bit che cade dalla parte estrema sinistra del moltiplicando. Questo bit cadrà nel bit carry. Esso qui non può essere immagazzinato permanentemente perchè poi può essere distrutto da qualsiasi operazione aritmetica. Questo dovrebbe essere conservato in un registro "permanente". Esso dovrebbe essere fatto scorrere nella locazione di memoria TMP. Questo è infatti realizzato dall'istruzione successiva:

ROL TMP

Questa specifica: "Rotazione a sinistra" dei contenuti di TMP.

Qui si può fare un'interessante osservazione. Sono stati appena utilizzati due diversi tipi di istruzioni di scorrimento per fare scorrere un registro di una posizione a sinistra. La prima è ASL. La seconda è ROL. Qual'è la differenza?

L'istruzione ASL fa scorrere i contenuti del registro. L'istruzione ROL è un'istruzione di rotazione. Essa sposta i contenuti del registro di una posizione a sinistra ed il bit che cade dall'estrema sinistra va nel bit carry, come al solito. La differenza sta nel fatto che i *contenuti precedenti del bit carry sono forzati nella posizione più a destra.* Questa in matematica è chiamata rotazione circolare (rotazione a 9 bit). Questo è esattamente quello che si vuole come risultato della ROL, il bit spinto fuori da TMP sulla sinistra e preservato nel bit carry C arriverà nella posizione più a destra del registro TMP. Così opera come si voleva.

Si è così terminato con la parte aritmetica di questo programma. Si dovrà verificare se l'operazione è stata eseguita otto volte, cioè se la moltiplicazione è terminata. Normalmente nella maggior parte dei microprocessori questo richiede due istruzioni:

DEX

Questa istruzione decrementa i contenuti del registro X. Se esso conteneva 8, dopo l'esecuzione di questa istruzione esso conterrà 7.

BNE MULT

Questa è un'altra istruzione di verifica e diramazione. Essa specifica "salta alla locazione MULT se il risultato non è uguale a 0". Finchè il registro contatore decrementa ad un intero non zero, c'è un salto automatico indietro alla label MULT. Questo è chiamato ciclo di moltiplica-

zione. Con riferimento al precedente diagramma di flusso questo corrisponde alla freccia che esce dall'ultimo blocco. Questo ciclo sarà eseguito 8 volte.

**Esercizio 3.11:** *Cosa succede quando X è decrementato a 0? Qual'è la successiva istruzione che viene eseguita?*

Nella maggior parte dei casi il programma appena sviluppato sarà una subroutine e l'istruzione finale della subroutine sarà RTS. Il meccanismo della subroutine sarà spiegato in seguito nel corso di questo capitolo.

## AUTO-TEST IMPORTANTE

Se si desidera imparare come programmare è estremamente importante capire un programma tipico nei dettagli completi. Sono state introdotte molte nuove istruzioni. L'algoritmo è ragionevolmente semplice ma il programma è molto più lungo dei programmi precedentemente sviluppati. *Si suggerisce vivamente di eseguire completamente e correttamente il seguente esercizio prima di procedere nel corso di questo capitolo.* Se si fa questo correttamente si avrà realmente capito il meccanismo mediante il quale le istruzioni manipolano i contenuti della memoria e dei registri del microprocessore e come deve essere utilizzato il flag carry. Se non si fa questo è probabile che si provino difficoltà nella scrittura da soli dei programmi. Si proceda quindi all'esecuzione del seguente esercizio.

**Esercizio 3.12:** *Ogni volta che viene scritto un programma si dovrebbe verificarlo manualmente in modo da accertare che i suoi risultati saranno corretti. Si farà proprio questo: lo scopo di questo esercizio è di riempire la tabella di Figura 3.12.*

Si può scrivere direttamente su questa ovvero su una sua copia. Lo scopo è determinare i contenuti di ogni registro e locazione di memoria di rilievo del sistema dopo l'esecuzione di ogni istruzione di questo programma, dall'inizio alla fine. Nella Figura 3.12 si troveranno riportati orizzontalmente tutti i registri e locazioni utilizzati dal programma: X, A, MPR, C (il bit di flag carry), TMP, MPD, RESL, RESH. Sulla parte sinistra dell'istruzione si deve riportare la label, se disponibile, e l'istruzione da eseguire. A destra dell'istruzione si devono scrivere i contenuti di ogni registro dopo l'esecuzione di questa istruzione. Ogni volta che i contenuti di un registro sono indefiniti si utilizzerà un tratto. Si inizia riempendo assieme questa tabella. Si dovrà poi riempirla fino alla fine.

La prima riga è la seguente:

| LABEL | ISTRUZIONE | X | A | MPR | C | TEMP | MPD | (RES)L | (RES)H |
|---|---|---|---|---|---|---|---|---|---|
| | LDA #0 | ------ | 00000000 | 00000011 | -- | ------ | 00000101 | ------ | ------ |

Figura 3.13: Prima istruzione della moltiplicazione

La prima istruzione da eseguire è LDA # 0.

Dopo l'esecuzione di questa istruzione, i contenuti del registro X non sono noti. Questo è indicato con trattini. I contenuti dell'accumulatore sono tutti zeri. Si assume anche che il moltiplicatore ed il moltiplicando siano stati caricati dal programmatore precedente l'esecuzione di questo programma. (Altrimenti sono necessarie istruzioni addizionali per posizionare i contenuti di MPR ed MPD). In MPR si trova il valore binario di "3". In MPD si trova il valore binario di "5". Il bit carry non è definito e così pure il registro TMP ed entrambi i registri utilizzati per RES. Si riempia ora la riga successiva. Essa è riportata di seguito: la sola differenza è che i contenuti del registro TMP sono stati posti a "0". L'istruzione successiva porrà a "0" i contenuti di RESAD e quella ancora successiva porrà a "0" i contenuti di RESAD + 1'

| LABEL | ISTRUZIONE | X | A | MPR | C | TEMP | MPD | (RES)L | (RES)H |
|---|---|---|---|---|---|---|---|---|---|
| | LDA #0 | ------ | 00000000 | 00000011 | -- | ------ | 00000101 | ------ | ------ |
| | STA TEMP | | | | | 00000000 | | | |

Figura 3.14: Prime due righe della moltiplicazione

La quinta istruzione: LDX # 8 porrà i contenuti di X ad "8". Si consideri un'ulteriore istruzione (vedere Figura 3-15).

L'istruzione LSR MPRAD farà scorrere i contenuti di MPRAD a destra di una posizione. Si può vedere che dopo lo scorrimento i contenuti di MPR sono "0000 0001". L'"1" più a destra di MPR è caduto nel bit carry. Il bit C è ora posto ad 1. Gli altri registri sono invariati.

Si proceda ora da soli: si riempia completamente il resto di questa tabella. Non è difficile ma questo richiede attenzione. Se si hanno dubbi sulle regole di alcune istruzioni, si può far riferimento al Capitolo 4 dove si può trovare ciascuna di queste elencate e descritte, oppure anche alla parte di Appendice di questo libro dove esse sono riportate in forma di tabella.

Il risultato finale della moltiplicazione dovrebbe essere "15" in forma binaria, contenuto nei registri RES basso ed alto. RES alto dovrebbe

essere posto a "0000 0000". RES basso dovrebbe essere posto a "0000 1111". Se è stato ottenuto questo risultato, l'esercizio è stato risolto correttamente. Diversamente si provi ancora una volta. La sorgente più frequente di errori è una manomissione del bit carry. Ci si assicuri che il bit carry sia cambiato ogni volta che si esegue un'istruzione aritmetica. Non si dimentichi che la ALU porrà il bit carry dopo ogni operazione di addizione.

| LABEL | ISTRUZIONE | X | A | MPR | C | TEMP | MPD | (RES)L | (RES)H |
|---|---|---|---|---|---|---|---|---|---|
| | LDA #0 | ------ | 00000000 | 00000011 | -- | ------ | 00000101 | ------ | ------ |
| | STA TEMP | | | | | 00000000 | | | |
| 000 | STA RESAD | | | | | | | 00000000 | |
| | STA RESAD + 1 | | | | | | | | 00000000 |
| | LDX #8 | 00001000 | | | | | | | |
| MULT | LSR MPRAD | | | 00000001 | 1 | | | | |
| | BCC NOADD | | | | | | | | |
| | LDA RESAD | | | | | | | | |
| | CLC | | | | 0 | | | | |
| | ADC MPDAD | | 00000101 | | | | | | |
| 101 | STA RESAD | | | | | | | 00000101 | |
| | LDA RESAD + 1 | | 00000000 | | | | | | |
| | ADC TEMP | | | | | | | | |
| | STA RESAD + 1 | | | | | | | | |
| NOADD | ASL MPDAD | | | | | | 00001010 | | |
| | ROL TEMP | | | | | | | | |
| | DEX | 00000111 | | | | | | | |
| | BNE MULT | | | | | | | | |
| MULT | LSR MPRAD | (2nd ITERATION) | | 00000000 | 1 | | | | |

Figura 3.15: Tabulato parzialmente completo per l'esercizio 3-12

## Alternative di Programmazione

Il programma appena sviluppato è solo uno dei molti modi in cui esso potrebbe essere scritto. Ogni programmatore può trovare modi per cambiare e talvolta migliorare un programma. Per esempio è stato spostato il moltiplicando a sinistra prima di sommare. Sarebbe stato matematicamente equivalente allo spostamento del risultato di una posizione a destra prima di sommarlo al moltiplicando. Il vantaggio è che non sarebbe richiesto il registro TMP, risparmiando così una locazione di memoria. Questo potrebbe essere un metodo preferito in un microprocessore equipaggiato con sufficienti registri interni cosicchè MPR, MPD e RES potrebbero essere contenuti all'interno del microprocessore. Poichè si è obbligati ad utilizzare la memoria per eseguire queste operazioni, il risparmio di una locazione di memoria non è di rilievo. Il punto è quindi se il secondo metodo può portare ad una moltiplicazione più veloce. Questo è un esercizio interessante:

**Esercizio 3.13:** *Si scriva una moltiplicazione di 8 $\times$ 8 bit impiegando lo stesso algoritmo ma facendo scorrere il risultato di una posizione a destra*

*invece di far scorrere il moltiplicando di una posizione a sinistra. Si confrontino i due programmi precedenti e si determini se questo diverso approccio potrebbe essere più veloce o più lento di quello precedente.*

Può sorgere un altro problema: per determinare la velocità del programma si può far riferimento alla tabella del paragrafo di Appendice che elenca il numero di cicli richiesti da ciascuna istruzione. Comunque il numero di cicli richiesti da alcune istruzioni dipende da dove esse sono localizzate. Esiste uno speciale modo di indirizzamento del 6502 chiamato Modo di *Indirizzamento Diretto* dove la prima pagina (locazioni da 0 a 255) è riservata all'esecuzione veloce. Questo sarà spiegato al Capitolo 5 sulle tecniche di indirizzamento. Brevemente tutti i programmi che richiedono un'esecuzione veloce saranno localizzati in pagina 0 cosicchè le istruzioni richiedono solo due byte per indirizzare le locazioni di memoria (l'indirizzamento di 256 locazioni richiede solo un byte), mentre le istruzioni localizzate in posizione generica nella memoria richiederanno tipicamente istruzioni di 3 byte. Quest'analisi verrà ripresa al Capitolo 5.

## Un Programma di Moltiplicazione migliorato

Il programma appena sviluppato è una traduzione diretta in codice dell'algoritmo. Comunque la programmazione effettiva richiede una stringente attenzione ai dettagli cosicchè la lunghezza del programma può essere ridotta per migliorare la sua velocità di esecuzione. Si presenterà ora una realizzazione migliorata dello stesso algoritmo.

Uno dei compiti che consuma istruzioni e tempo è lo scorrimento del risultato e del moltiplicatore. Un "espediente" convenzionale utilizzato nell'algoritmo della moltiplicazione è basato sull'osservazione seguente: ogni volta che il moltiplicatore è fatto scorrere di una posizione di bit a destra, diventa disponibile sulla sinistra una posizione di bit. Contemporaneamente si può osservare che il primo risultato (o prodotto parziale) utilizzerà, al più, 9 bit. Dopo il successivo scorrimento di moltiplicazione, la dimensione del prodotto parziale aumenterà ancora di un bit. In altre parole si può riservare, inizialmente, una locazione di memoria per il prodotto parziale e poi utilizzare la posizione di bit che è stata liberata dal moltiplicatore in virtù del suo scorrimento.

Si sta ora facendo scorrere il moltiplicatore a sinistra. Si libererà una posizione di bit sulla destra. Si fa entrare il bit più a destra del prodotto parziale in questa posizione di bit appena liberata. Si consideri ora il programma.

Si consideri anche l'utilizzazione ottima dei registri. I registri interni del 6502 appaiono in Figura 3-16. X è meglio utilizzato come un contatore. Questo sarà utilizzato per contare il numero di bit spostati. L'accumulatore (sfortunatamente) è il solo registro interno che può essere fatto scorrere. Per migliorare l'efficienza, si dovrebbe immagazzinare in esso il moltiplicatore oppure anche il risultato.

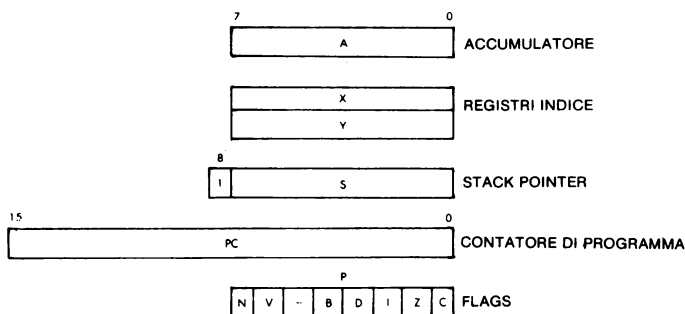



Figura 3.16: I registri del 6502

Quale si metterà nell'accumulatore? Il risultato deve essere sommato al moltiplicando ogni volta che scorre fuori un 1. Poichè il 6502 somma sempre soltanto qualcosa all'accumulatore, è il risultato che risiederà nell'accumulatore.

Gli altri numeri devono risiedere nella memoria (vedere Figura 3-17).

A e B conserveranno il risultato. A conserverà la parte alta del risultato e B quella bassa. A è l'accumulatore e B una locazione di

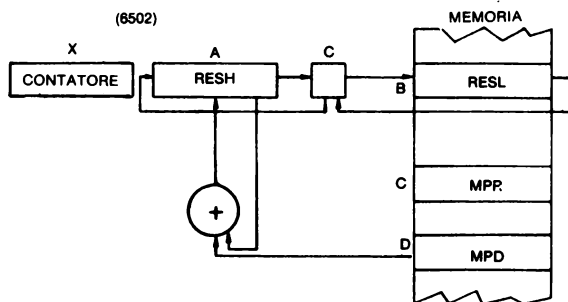



Figura 3.17: Allocazione dei registri (moltiplicazione migliorata)

memoria, preferibilmente in pagina 0. C conserverà il moltiplicatore (una locazione di memoria). D conserva il moltiplicando (una locazione di memoria). Il programma risulta quindi:

```
MULT    LDA  # 0    INIZIALIZZA IL RISULTATO A ZERO (ALTO)
        STA  B      INIZIALIZZA IL RISULTATO (BASSO)
        LDX  # 8    X È IL CONTATORE DI SCORRIMENTI
LOOP    LSR  C      SCORRE MPR
        BCC  NOADD
        CLC         CARRY ERA UNO. VIENE AZZERATO
        ADC  D      A = A + MPD
NOADD   ROR  A      SCORRE IL RISULTATO
        ROR  B      BIT INSERITO IN B
        DEX         DECREMENTA IL CONTATORE
        BNE  LOOP   ULTIMO SCORRIMENTO?
```

Figura 3.18: Moltiplicazione migliorata

Si esamini il programma. Poichè A e B conservano il risultato e devono essere inizializzati al valore 0. Questo viene eseguito da:

```
MULT LDA # 0
     STA B
```

Si utilizzerà quindi il registro X come contatore di scorrimento e sarà inizializzato al valore 8:

```
LDX # 8
```

Si è ora pronti per entrare nel ciclo di moltiplicazione principale come in precedenza. Si farà scorrere prima il moltipliatore, quindi si verificherà il bit carry che conserva il bit più a destra del moltiplicatore caduto fuori. Operano questo:

```
LOOP LSR C
BCC NOADD
```

Qui si fa scorrere il moltiplicatore a sinistra (invece che prima a destra). Questo è equivalente al precedente algoritmo perchè l'operazione di addizione è commutativa.

Esistono due possibilità: se il carry era 0 si andrà a NOADD. Si

assuma che il carry sia 1. Si procederà:

```
        CLD
        ADC D
```

Poichè il Carry era 1, esso deve essere azzerato e quindi sommare il moltiplicando all'accumulatore. (L'accumulatore conserva i risultati, 0 fin'ora).
Si faccia ora scorrere il prodotto parziale:

```
NOADD ROR A
      ROR B
```

Il prodotto parziale in A è fatto scorrere a destra di un bit. Il bit più a destra cade nel bit carry. Il bit cary è catturato e ruotato nel registro B, che conserva la parte bassa del risultato.
Si deve ora verificare semplicemente se l'operazione è conclusa:

```
        DEX
        BNE LOOP
```

Se si esamina questo nuovo programma risulta che è formato da un numero di istruzioni circa metà di quello precedente. Esso sarà anche eseguito molto più velocemente. Questo mostra l'importanza del selezionamento corretto dei registri che contengono l'informazione.
Un progetto diretto originerà un programma che lavora. Ma non originerà un programma ottimizzato. Perciò è molto importante utilizzare i registri disponibili e le locazioni di memoria nel modo migliore possibile. Questo esempio illustra un approccio razionale alla selezione dei registri per ottenere la massima efficienza.

**Esercizio 3.14:** *Si calcoli la velocità di un'operazione di moltiplicazione utilizzando quest'ultimo programma. Si assuma che una diramazione si verifichi nel quindici per cento dei casi. Si ricavi il numero di cicli richiesti da ogni istruzione nella tabella alla fine del libro. Si assuma una velocità di clock con un ciclo = 1 microsecondo.*

## Divisione Binaria

L'algoritmo per la divisione binaria è analogo a quello utilizzato per la moltiplicazione. Il divisore è successivamente sottratto dai bit di ordine elevato del dividendo. Dopo ogni sottrazione, il risultato è utilizzato al posto del dividendo iniziale. Il valore del quoziente è contemporaneamente aumentato di 1 ogni volta. Eventualmente il risultato della sottra-

zione è negativo. Questo è chiamato un eccesso. Si deve quindi immagazzinare il risultato parziale riaggiungendo il divisore ad esso. Naturalmente il quoziente deve essere contemporaneamente decrementato di 1. Il quoziente e dividendo sono poi fatti scorrere di una posizione di bit e l'algoritmo è ripetuto.

Il metodo appena descritto è chiamato *metodo a ri-immagazzinamento.* Una variazione di questo metodo che produce un miglioramento di velocità di esecuzione è detto metodo *senza ri-immagazzinamento.*

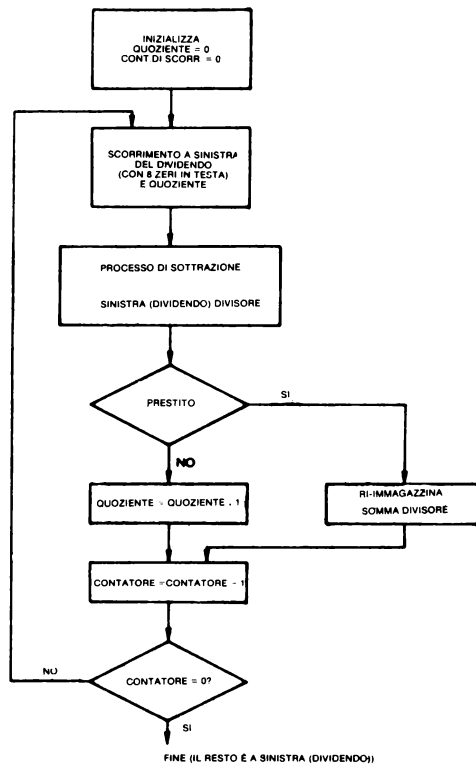



Figura 3.19: Diagramma di flusso della divisione binaria ad 8 bit

## La divisione a 16 bit

Verrà ora descritta la divisione senza rimemorizzazione per un dividendo a 16 bit ed un divisore di 8 bit. La Fig. 3-20 riporta il registro e la

locazione di memoria di questo programma. Il dividendo è contenuto nell'accumulatore (parte alta) e nella locazione di memoria 0, qui indicata con B. Il risultato è contenuto in Q (locazione di memoria 1). Il divisore è contenuto in D (locazione di memoria 2). Il risultato sarà contenuto in D ed A (A conterrà il resto).

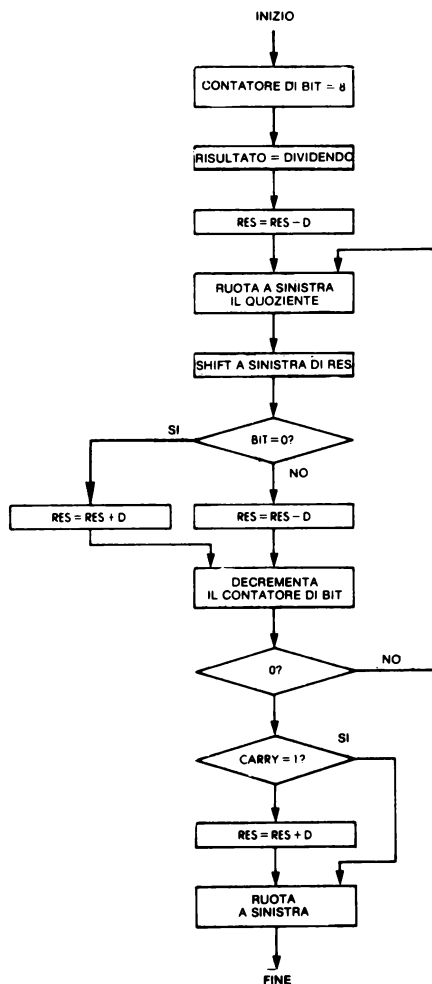



Figura 3.20: Diagramma di flusso divisione 16x8

La Fig. 3-21 riporta il programma, mentre il diagramma di flusso corrispondente è riportato in Fig. 3-22.

```
LINE #  LOC    CODE       LINE

0002    0000                       * = $0
0003    0000              B        * = * + 1
0004    0001              Q        * = * + 1
0005    0002              D        * = * + 1
0006    0003                       * = $200
0007    0200   A0 08      DIV      LDY #8
0008    0202   38                  SEC
0009    0203   E5 02               SBC D
0010    0205   08         LOOP     PHP
0011    0206   26 01               ROL Q
0012    0208   06 00               ASL B
0013    020A   2A                  ROL A
0014    020B   28                  PLP
0015    020C   90 05               BCC ADD
0016    020E   E5 02               SBC D
0017    0210   4C 15 02            JMP NEXT
0018    0213   65 02      ADD      ADC D
0019    0215   88         NEXT     DEY
0020    0216   D0 ED               BNE LOOP
0021    0218   B0 03               BCS LAST
0022    021A   65 02               ADC D
0023    021C   18                  CLC
0024    021D   26 01      LAST     .ROL Q
0025    021F   00                  BRK
0026    0220                       .END
```

Figura 3.21: Programma

**Esercizio 3-15:** *Si verifichi il funzionamento corretto di questo programma eseguendo la divisione a mano e verificando il programma in modo analogo all'Esercizio 3-12. Si divida 33 per 3. Il risultato, naturalmente, dovrebbe essere 11 con resto 0.*

## OPERAZIONI LOGICHE

L'altra classe di istruzioni che la ALU può eseguire all'interno del microprocessore è il set di istruzioni logiche. Queste comprendono AND, OR ed OR esclusivo (EOR). Inoltre si possono comprendere qui anche le operazioni di scorrimento che sono già state utilizzate e l'istruzione di confronto chiamata CMP per il 6502. L'impiego singolo di AND, ORA, EOR sarà descritto al Capitolo 4 sul set di istruzioni del 6502. Si svilupperà ora un breve programma che controllerà se una data

locazione di memoria chiamata LOC contiene il valore "0", il valore "1" oppure qualcos'altro. Il programma è il seguente:

```
                    LDA   LOC    LEGGE CARATTERE IN LOC
                    CMP   # $00  CONFRONTA CON ZERO
                    BEQ   ZERO   È UNO ZERO?
                    CMP   # $01  1?
                    BEQ   ONE
TROVATO NIENTE      ...
                    ...
ZERO                ...
                    ...
ONE                 ...
```

La prima istruzione: LDA LOC legge i contenuti della locazione di memoria LOC. Questo è il carattere che si vuole provare.

CMP # $00

Questa istruzione confronta i contenuti dell'accumulatore col valore esadecimale letterale "00" cioè con la struttura di bit "00000000". Questa istruzione di confronto porrà il bit Z del registro dei flag, che sarà poi controllato dall'istruzione successiva.

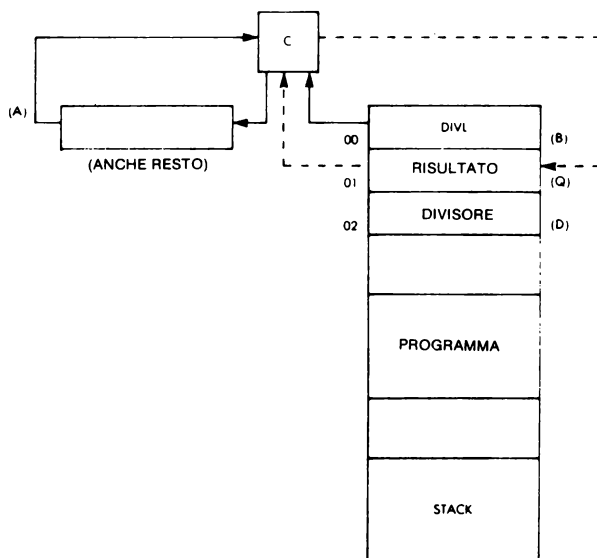



Figura 3.22: Diagramma di flusso della divisione 16x8 (senza rimemorizzazione del risultato ad 8 bit)

BEQ ZERO

L'istruzione BEQ specifica "diramazione se uguale". L'istruzione di diramazione determinerà se la verifica è soddisfatta esaminando il bit Z. Se sì il programma salterà a ZERO. Se il test non è soddisfatto allora viene eseguita l'istruzione successiva in ordine sequenziale:

CMP # $01

Il processo sarà ripetuto per un'altra struttura. Se il test è verificato l'istruzione successiva risulterà da un salto alla locazione uno. Se fallisce viene eseguita l'istruzione successiva in ordine sequenziale.

**Esercizio 3.16:** *Si scriva un programma che legga i contenuti della locazione di memoria "24" e salti all'indirizzo chiamato "STAR" se c'era un "*" nella locazione di memoria 24. La struttura di bit per un "*" nella notazione in linguaggio assembly è rappresentato da "00101010".*

### Sommario

Sono state ora studiate le istruzioni più importanti del 6502 mediante la loro utilizzazione diretta. I valori sono stati trasferiti tra la memoria ed i registri. Sono state eseguite operazioni aritmetiche e logiche su tali dati. Sono state verificate ed, in dipendenza del risultato di questo test, sono state eseguite varie porzioni di programma. È stata anche introdotta una struttura chiamata ciclo nel programma della moltiplicazione. Verrà ora introdotta un'importante struttura della programmazione: la Subroutine.

## SUBROUTINE

Concettualmente una subroutine è semplicemente un blocco di istruzioni alle quali è stato assegnato un nome dal programmatore. Da un punto di vista pratico, una subroutine deve iniziare con una speciale istruzione chiamata la dichiarazione della subroutine che la identifica per l'assemblatore. Inoltre deve terminare con un'altra speciale istruzione chiamata *ritorno.* Innanzi tutto si illustrerà l'uso delle subroutine nel programma in modo da illustrarne l'importanza. Quindi si esaminerà come esse sono effettivamente realizzate.

L'impiego di una subroutine è illustrato in Figura 3.23. Il programma principale appare sulla sinistra dell'illustrazione. La subroutine è rappresentata simbolicamente sulla destra. Si esamini il meccanismo della subroutine. Le righe del programma principale sono successivamente eseguite finchè non si incontra una nuova istruzione di chiamata "SUB". Questa istruzione speciale è una *chiamata di subroutine* e si risolve in un

trasferimento dell'esecuzione alla subroutine. Questo significa che l'istruzione successiva da eseguire dopo la CALL SUB è la prima istruzione all'interno della subroutine. Questo è illustrato dalla freccia 1 nell'illustrazione.

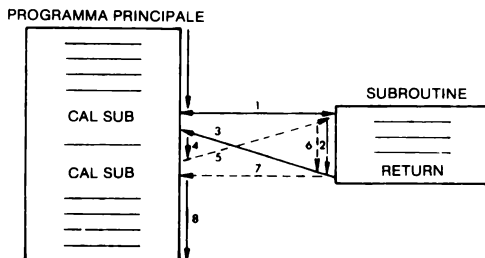



Figura 3.23: Chiamate di subroutine

Quindi il sottoprogramma all'interno della subroutine viene eseguito proprio come qualsiasi altro programma. Si assumerà che la subroutine non contenga nessun'altra chiamata. L'ultima istruzione di questa subroutine è un RETURN. Questa è un'istruzione speciale che originerà un ritorno al programma principale. L'istruzione successiva da eseguire dopo RETURN è quella seguente la CALL SUB. Questo è mostrato dalla freccia 3 nell'illustrazione. L'esecuzione del programma continua quindi come illustrato dalla freccia 4.

Nel corpo del programma principale appare una seconda CALL SUB. Si verifica un nuovo trasferimento, mostrato dalla freccia 5. Questo significa che il corpo della subroutine è ancora eseguito successivamente all'istruzione CALL SUB.

Ogni volta che si incontra RETURN all'interno della subroutine si verifica un ritorno all'istruzione successiva la CALL SUB in questione. Questo è illustrato dalla freccia 7. In seguito al ritorno al programma principale, l'esecuzione del programma procede normalmente, come illustrato dalla freccia 8.

Il ruolo delle due istruzioni speciali CALL SUB e RETURN è così chiarito. Qual'è l'importanza della subroutine?

L'importanza essenziale della subroutine è che essa può essere richiamata da un qualsiasi numero di punti del programma principale ed utilizzata ripetutamente senza la sua riscrittura. Un primo vantaggio è che questo approccio risparmia spazio di memoria e non c'è necessità di riscrivere la subroutine ogni volta. Un secondo vantaggio è che il pro-

grammatore può progettare una subroutine specifica solo una volta e quindi usarla ripetutamente. Questo è una semplificazione significativa del progetto del programma.

**Esercizio 3.17:** *Qual'è il principale svantaggio di una subroutine?*

Lo svantaggio di una subroutine potrebbe essere chiaro proprio dall'esame del flusso di esecuzione tra il programma principale e la subroutine. Una subroutine si risolve in una esecuzione più lenta poichè devono essere eseguite ulteriori istruzioni: la CALL SUB ed il RETURN.

## Realizzazione del Meccanismo della Subroutine

Si esaminerà qui come le due speciali istruzioni CALL SUB e RETURN, sono realizzate all'interno del processore. L'effetto dell'istruzione CALL SUB è di causare il prelievo dell'istruzione successiva ad un nuovo indirizzo. Si ricorderà (altrimenti si rilegga il Capitolo 1) che l'indirizzo dell'istruzione successiva da eseguire in un calcolatore è contenuto nel contatore di programma (PC). Questo significa che l'effetto della CALL SUB è la sostituzione di nuovi contenuti nel registro PC. Il suo effetto è di caricare l'indirizzo iniziale della subroutine nel contatore di programma. *Questo è in realtà sufficiente?*

Per rispondere a questa domanda si consideri l'altra istruzione che deve essere realizzata: il RETURN. Il RETURN deve originare, come indica il suo nome, un ritorno all'istruzione che segue la CALL SUB. Questa è possibile solo se l'indirizzo di questa istruzione è stato preservato da qualche parte. Questo indirizzo deve essere il valore del contatore di programma all'istante in cui si incontra la CALL SUB. Questo perchè il contatore di programma è incrementato automaticamente ogni volta che viene utilizzato (vedere Capitolo 1). Questo è precisamente l'indirizzo che si vuole preservare così da poter successivamente eseguire il RETURN.

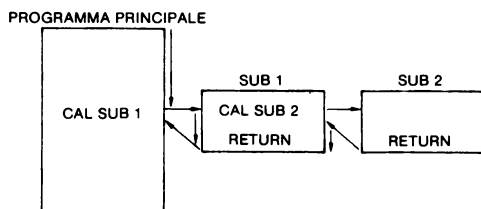



Figura 3.24: Chiamate annidate

Il problema successivo è: dove si può conservare questo indirizzo di

ritorno? Questo indirizzo deve essere conservato in una locazione ragionevole dove è sicuro che non sarà cancellato. Comunque si consideri ora la situazione seguente, illustrata dalla Figura 3-24: in questo esempio la subroutine 1 contiene una chiamata a SUB 2. Il meccanismo potrebbe lavorare correttamente in questo caso. Naturalmente possono esserci molte più di due subroutines, dette N chiamate "annidate". Ogni volta che si incontra una nuova CALL il meccanismo deve perciò immagazzinare ancora il contatore di programma. Questo implica la necessità di almeno 2N locazioni di memoria per questo meccanismo. Addizionalmente sarà necessario ritornare da SUB 2 prima e SUB 1 poi. In altre parole è necessaria una struttura che possa preservare l'ordine cronologico in cui i dati devono essere conservati.

La struttura ha un nome. È già stata introdotta. È *lo stack*. La figura 3-26 mostra i contenuti effettivi dello stack durante le chiamate di subroutine successive. Si osservi prima il programma principale. All'indirizzo 100 si incontra la prima chiamata: CALL SUB 1. Si assumerà che, in questo processore, la chiamata di subroutine utilizzi 3 byte. L'indirizzo sequenzialmente successivo non è perciò "101" ma "103". L'istruzione di chiamata utilizza gli indirizzi "100", "101", "102". Poichè l'unità di controllo del 6502 "sa" che si tratta di un'istruzione di 3 byte, il valore del contatore di programma quando la chiamata è stata completamente decodificata sarà "103". L'effetto della chiamata sarà di caricare il valore "280" nel contatore di programma. "280" è l'indirizzo di partenza di SUB 1.

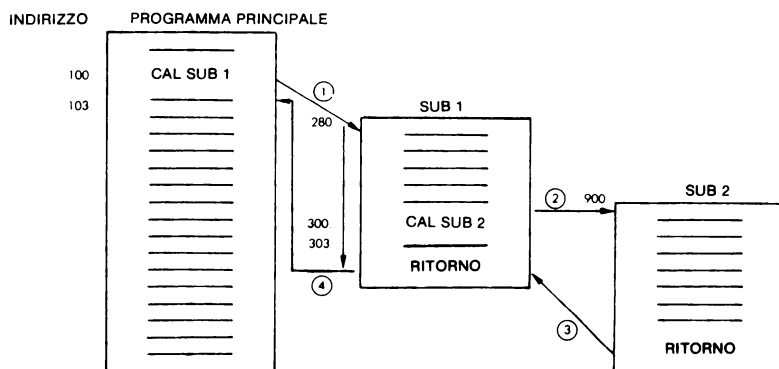



Figura 3.25: Le chiamate di subroutine

Il secondo effetto della CALL sarà di spingere nello stack (per preservare) il valore "103" del contatore di programma. Questo è illustrato

nella parte destra in basso dell'illustrazione. Alla locazione 300 si incontra una nuova chiamata. Analogamene al caso precedente il valore "900" sarà caricato nel contatore di programma. Questo è l'indirizzo di partenza della SUB 2. Contemporaneamente il valore "303" sarà spinto nello stack. Questo è mostrato in basso a sinistra nell'illustrazione dove l'ingresso all'istante 2 è "303". L'esecuzione procederà quindi a destra dell'illustrazione all'interno di SUB 2.

Si è ora pronti per dimostrare l'effetto dell'istruzione RETURN e per il funzionamento corretto del meccanismo dello stack. L'esecuzione procede all'interno di SUB 2 finchè non si incontra l'istruzione RETURN all'istante 3. L'effetto dell'istruzione RETURN è semplicemente quello di far uscire la sommità dello stack inviandola nel contatore di programma. In altre parole il contatore di programma è riimmagazzinato al suo valore precedente l'ingresso nella subroutine. La sommità dello stack nell'esempio è "303". La figura 3-26 mostra che, all'istante 3, il valore "303" è stato rimosso dallo stack e riposizionato nel contatore di programma. Come risultato l'esecuzione di istruzioni procede dall'indirizzo "303". All'istante 4 si incontra il RETURN di SUB 1. Il valore alla sommità dello stack è "103". Esso viene prelevato e portato nel contatore di programma. Come risultato l'esecuzione del programma procederà dalla locazione "103" all'interno del programma principale. Questo è proprio l'effetto che si voleva. La Figura 3-26 mostra che all'istante 4 lo stack è nuovamente vuoto. Quindi il meccanismo funziona.

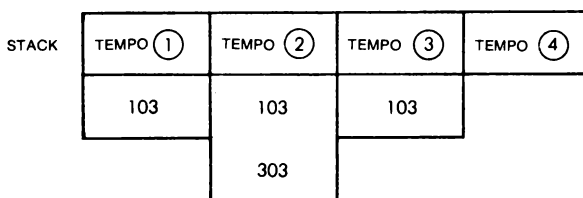



Figura 3.26: Lo stack in funzione del tempo

Il meccanismo di chiamata di subroutine funziona fino alla massima dimensione dello stack. Questa è la ragione per cui i primi microprocessori che avevano uno stack di 4 od 8 registri erano essenzialmente limitati a 4 od 8 livelli di chiamata di subroutine. In teoria il 6502, che ha uno stack limitato a 256 locazioni di memoria (Pagina 1), può accomandare fino a 256 successive chiamate di subroutine. Questo è vero solo se non ci sono interrupt, se lo stack non viene utilizzato per nessun altro scopo e

se nessun registro necessita di essere memorizzato all'interno dello stack. In pratica vengono utilizzati pochi livelli di subroutine.

Si noti che, nelle illustrazioni 3-24 e 3-25, le subroutine sono state indicate a destra del programma principale. Questo è solo per chiarezza del diagramma. In realtà le subroutine sono impostate dall'utente come normali istruzioni del programma. Su un foglio di carta, ovvero la lista di un programma completo, le subroutine possono essere all'inizio del testo, a metà, oppure alla fine. Questo perchè esse sono precedute da una dichiarazione di subroutine: esse devono essere identificate. le istruzioni speciali dicono all'assemblatore che quello che segue deve essere trattato come una subroutine. Tali *direttive* dell'assemblatore saranno presentate al Capitolo 9.

### Subroutine del 6502

È stato ora descritto il meccanismo della subroutine e come lo stack viene impiegato per realizzarlo. L'istruzione di chiamata di subroutine per il 6502 è detta JSR (salta alla subroutine). Questa è proprio un'istruzione a 3 byte. Sfortunatamente questo è un salto incondizionato: non esistono dei bit di prova. Occorre inserire una diramazione esplicita prima di JSR se deve essere eseguito un test.

Il ritorno da subroutine è l'istruzione RTS (ritorno da subroutine). Questa è un'istruzione di 1 byte.

**Esercizio 3-18:** *Perchè il ritorno da una subroutine è molto più veloce della chiamata? (Suggerimento: se la risposta non è ovvia si osservi ancora la realizzazione dello stack del meccanismo della subroutine e si analizzino le operazioni interne che devono essere eseguite).*

### Esempi di Subroutine

La maggior parte dei progammi da sviluppare e che si sviluppano potrebbero essere normalmente scritti come subroutine. Per esempio il programma della moltiplicazione potrebbe essere utilizzato da molte aree del programma. Per facilitare lo sviluppo del programma e per motivi di chiarezza, è perciò conveniente definire una subroutine il cui nome sia per esempio MULT. Alla fine di questa subroutine si dovrebbe aggiungere semplicemente l'istruzione RTS.

**Esercizio 3.19:** *Se MULT è utilizzato come subroutine, si potrebbe "danneggiare" qualsiasi flag o registro interno?*

**Recursione**

Recursione è una parola utilizzata per indicare che una subroutine sta chiamando se stessa. Se è stato capito il meccanismo si dovrebbe essere in grado di rispondere alle seguenti domande:

**Esercizio 3.20:** *È giusto che una subroutine chiami se stessa? (In altre parole, lavorerà sempre anche se una subroutine chiama se stessa? Se non si è sicuri si disegni lo stack e lo si riempia con gli indirizzi successivi. Si verificherà fisicamente se esso lavora oppure no. Questo risponderà alla domanda se il meccanismo lavora. Quindi si osservino i registri e la memoria e si determini se esiste un problema.*

**Parametri della Subroutine**

Quando si chiama una subroutine, normalmente ci si aspetta che la subroutine lavori su alcuni dati. Per esempio nel caso della moltiplicazione si vuole trasmettere due numeri alla subroutine che eseguirà la moltiplicazione. Si vede nel caso della routine della moltiplicazione che questa subroutine si aspetta di trovare il moltiplicando ed il moltiplicatore in assegnate locazioni di memoria. Questo illustra il primo metodo di passaggio di parametri: attraverso la memoria. Sono usate altre due tecniche ed i parametri possono essere passati in tre modi:

1. Attraverso i registri
2. Attraverso la memoria
3. Attraverso lo stack

— I *registri* possono essere utilizzati per passare i parametri. Questa può essere una soluzione vantaggiosa, supponendo che i registri siano disponibili, poichè non è necessario utilizzare una locazione di memoria prefissata. La subroutine rimane quindi indipendente dalla memoria. Se viene utilizzata una locazione di memoria prefissata, qualsiasi altro utente di subroutine deve essere molto attento per utilizzare la stessa conversione e che la locazione di memoria sia davvero disponibile (si osservi il precedente Esercizio 3.19). Questo perchè, in molti casi, un blocco di locazioni di memoria è riservato semplicemente per passare i parametri tra le varie subroutine.
— *L'utilizzazione della memoria* ha il vantaggio di maggiore flessibilità (più dati) ma si risolve in minor adempimento e conduce a legare la subroutine ad una data area di memoria.
— Il deposito di parametri nello *stack* ha lo stesso vantaggio dell'utilizzazione dei registri: è indipendente dalla memoria. La subroutine semplice-

mente conosce che deve ricevere due parametri immagazzinati alla sommità dello *stack*. Naturalmente questo vantaggio ha uno svantaggio: si fa confusione introducendo dati nello stack e perciò si riduce il numero di livelli possibili di chiamata di subroutine. La scelta è lasciata al programmatore. Nel caso generale si desidera rimanere indipendenti dalle locazioni di memoria effettive il più possibile.

Se i registri non sono disponibili, la miglior soluzione successiva è normalmente l'impiego dello stack. Comunque se è necessario trasmettere alla subroutine una grande quantità di informazioni occorrerà utilizzare la memoria. Un modo elegante per aggirare il problema del passaggio di blocchi di dati è di trasmettere semplicemente un puntatore dell'informazione. Un *puntatore* (pointer) è l'indirizzo all'inizio del blocco. Un puntatore può essere trasmesso in un registro (nel caso del 6502, questo limita il puntatore ad 8 bit), od anche, nello stack (due locazioni dello stack possono essere utilizzate per immagazzinare un indirizzo a 16 bit).

Infine se nessuna delle due soluzioni è applicabile allora si può trovare un compromesso ritenendo che i dati si trovino in qualche locazione di memoria prefissata (la "cassetta-postale").

**Esercizio 3.21:** *Quale dei tre metodi precedenti è il migliore per la recursione?*

## Biblioteca di Subroutine

C'è un grosso vantaggio nella strutturazione di parti di un programma in subroutine identificabili: esse possono essere collaudate indipendentemente e possono avere un nome mnemonico. Poichè esse possono essere utilizzate in altre aree del programma, divengono condivisibili e si può quindi costruire una biblioteca di subroutine di utilità immediata. Comunque non esiste una panacea generale nella programmazione del calcolatore.

L'impiego sistematico di subroutine per qualsiasi gruppo di istruzioni che possono essere raggruppate da una funzione può anche risolversi in una scarsa efficienza. Il programmatore accorto dovrà soppesare i vantaggi in funzione degli svantaggi.

## SOMMARIO

Questo capitolo ha presentato il modo in cui l'informazione è manipolata mediante istruzioni all'interno del 6502. Sono stati introdotti algoritmi di complessità crescente e tradotti in programmi. Sono stati utilizzati i principali tipi di istruzioni.

Sono state inoltre definite strutture importanti come cicli, stack e subroutine.

Si dovrebbe ora aver acquisito una comprensione di base alla programmazione e le principali tecniche utilizzate nelle applicazioni convenzionali. Si studieranno ora le istruzioni disponibili.

CAPITOLO 4

# IL SET DI ISTRUZIONI DEL 6502

## PARTE I - DESCRIZIONE GLOBALE

### INTRODUZIONE

Questo capitolo analizzerà innanzitutto le varie classi di istruzione che sarebbero disponibili in un calcolatore general purpose. Si analizzeranno quindi una ad una tutte le istruzioni disponibili per il 6502 e si spiegherà in dettaglio il loro scopo ed il modo in cui esse influenzano i flag o possono essere utilizzate in relazione a vari modi di indirizzamento. Una discussione dettagliata delle tecniche di indirizzamento sarà presentata al Capitolo 5.

### CLASSI DI ISTRUZIONE

Le istruzioni possono essere classificate in molti modi e non esistono convenzioni. Si distingueranno qui cinque categorie principali di istruzioni:

1. trasferimento di dati
2. elaborazione di dati
3. test e diramazione
4. ingresso/uscita
5. controllo

Si esaminerà in dettaglio ciascuna di queste classi di istruzioni.

#### Trasferimento Dati

Le istruzioni di trasferimento dati trasferiranno i dati ad 8 bit tra due registri, oppure tra un registro e la memoria, ovvero tra un registro ed un dispositivo d'ingresso/uscita. Istruzioni di trasferimento specializzate possono esistere per registri che giocano un ruolo specifico. Per esempio:

un funzionamento di introduzione ad estrazione per un'efficiente realizzazione dello stack. Queste muoveranno una parola di dati tra la sommità dello stack e l'accumulatore in una istruzione singola mentre si ha l'aggiornamento automatico del registro puntatore dello stack.

## Elaborazione Dati

Le istruzioni di elaborazione dati si dividono in quattro categorie generali:

- operazioni aritmetiche (come più/meno)
- operazioni logiche (come AND, OR, OR esclusivo)
- operazioni di posizionamento e scorrimento (come scorrimento, rotazione, scambio)
- incremento e decremento

Si potrebbe notare che per un'efficiente elaborazione dati è desiderabile aver una potente costruzione aritmetica come moltiplicazione e divisione. Sfortunatamente questo non è disponibile sulla maggior parte dei microprocessori. È anche desiderabile avere potenti istruzioni di scorrimento e posizionamento, come lo spostamento di n bit, ovvero uno scambio di nibble, dove vengono scambiati la metà destra e quella sinistra di un byte. Queste non sono normalmente disponibili sulla maggior parte di microprocessori.

Prima di esaminare le effettive istruzioni del 6502 si richiama la differenza tra uno *scorrimento* e una *rotazione.* Lo scorrimento muoverà i contenuti di un registro o di una locazione di memoria, di una posizione di bit a destra o sinistra. Il bit che esce dal registro andrà nel bit carry. Il bit che entra dall'altra parte sarà uno "0".

Nel caso di una rotazione il bit che esce va ancora nel carry. Comunque il bit che entra è il precedente valore del bit carry. Questo corrisponde ad una rotazione a 9 bit. Potrebbe essere spesso desiderabile avere una vera rotazione ad 8 bit dove il bit che entra da una parte è questo che esce dall'altra. Questo non è normalmente disponibile sulla maggior parte di microprocessori. Infine nello scorrimento di una parola a destra è conveniente avere più tipi di scorrimento chiamati un'estensione di segno ovvero uno "spostamento aritmetico a destra". Nelle operazioni con numeri in complemento a 2, specialmente nella realizzazione di routine a virgola mobile, è spesso necessario spostare a destra un numero negativo. Quando si fa scorrere un numero in complemento a 2 a destra, il bit che deve entrare dalla parte sinistra dovrebbe essere 1 (il bit segno dovrebbe essere ripetuto tante volte quanto richiesto dagli scorri-

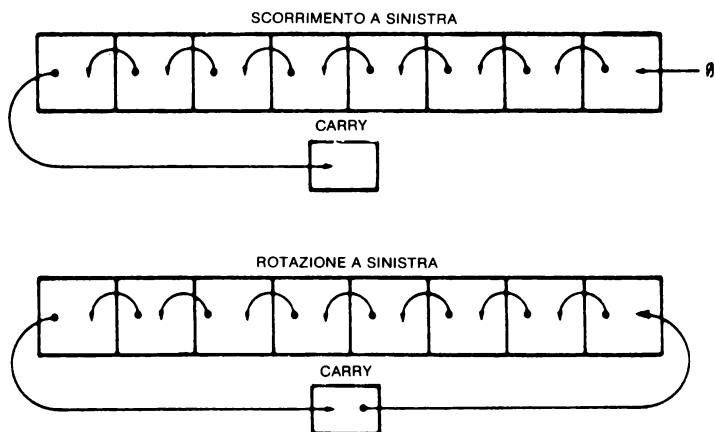



Figura 4.1: Scorrimento e rotazione

menti successivi. Sfortunatamente questo tipo di scorrimento non esiste nel 6502. Esso esiste in altri microprocessori.

**Test e Diramazione**

L'istruzione di test verificherà se tutti i bit del registro dei flag sono "0" od "1" oppure combinazioni di questi. Quindi è desiderabile avere più flag possibile in questo registro. Inoltre occorre essere in grado di verificare *qualsiasi posizione di bit all'interno di qualsiasi registro* e di verificare il contenuto di un registro rispetto al valore di qualunque altro (maggiore, minore oppure uguale a). Le istruzioni di test del microprocessore sono normalmente limitate alla verifica dei singoli bit del registro dei flag.

Le istruzioni di salto possono essere generalmente disponibili in tre categorie:

- il salto vero e proprio ad uno specificato indirizzo a 16 bit,
- la diramazione che spesso è ristretta ad un campo di spostamento di 8 bit,
- la chiamata che viene utilizzata con le subroutine.

È conveniente avere diramazioni a due oppure anche tre vie, in dipendenza, per esempio, se il risultato del confronto è "maggiore di", "minore di" oppure "uguale". È anche conveniente avere operazioni di salto che trasferiscono l'esecuzione in altri punti del programma. Infine,

nella maggior parte dei cicli, c'è un'operazione finale di decremento od incremento, seguita da un test ed una diramazione. La disponibilità di una singola istruzione di incremento/decremento più test e diramazione è perciò un vantaggio significativo per l'efficienza della realizzazione del ciclo. Questo non è disponibile nella maggior parte dei microprocessori. Sono disponibili soltanto diramazioni semplici, combinate con semplici test. Questo naturalmetne complica la programmazione e riduce l'efficienza.

### Ingresso/Uscita

Le istruzioni d'ingresso/uscita sono specializzate per la manipolazione di dispositivi ingresso/uscita. In pratica quasi tutti i microprocessori impiegano la *mappa-memoria I/O.* Questo significa che i dispositivi d'ingresso/uscita sono connessi al bus indirizzo proprio come chip di memoria ed indirizzati come tali. Essi appaiono al programmatore come locazioni di memoria. Tutte le operazioni tipiche della memoria possono essere applicate al dispositivo richiesto. Questo è vantaggioso per fornire una grande varietà di istruzioni che possono essere applicate. Lo svantaggio è che le operazioni tipiche della memoria normalmente richiedono 3 byte e sono perciò lente. In queste condizioni per un'efficiente manipolazione ingresso/uscita, è desiderabile avere disponibile un meccanismo di indirizzamento corto cosicchè i dispositivi I/O con velocità di manipolazione critica possano risiedere in Pagina 0. Comunque se è disponibile l'indirizzamento in Pagina 0, questo viene normalmente impiegato per la memoria RAM e perciò previene l'effettivo impiego per i dispositivi ingresso/uscita.

### Istruzioni di Controllo

Le istruzioni di controllo forniscono i segnali di sincronismo e possono sospendere oppure interrompere un programma. Esse possono anche funzionare come un break oppure un interrupt simulato. (Gli interrupt saranno descritti al Capitolo 6 sulle Tecniche d'Ingresso/Uscita).

## ISTRUZIONI DISPONIBILI SUL 6502

### Istruzioni di Trasferimento Dati

Il 6502 ha un set completo di istruzioni di trasferimento dati, eccetto che per il caricamento del puntatore dello stack che è ristretto in flessibilità.

I contenuti dell'accumulatore possono essere cambiati con una locazione di memoria con l'istruzione LDA (Carica) e STA (Immagazzina). Le stesse istruzioni si applicano ai registri X e Y. Queste sono rispettivamente le istruzioni LDX LDY ed STX STY. Non c'è invece un caricamento diretto per S. Vengono naturalmente forniti i trasferimenti tra registri: le istruzioni sono: TAX (trasferimento da A ad X), TAY, TSX, TXA, TXS, TYA. C'è una leggera asimmetria poichè i contenuti dello stack possono essere scambiati con X ma non con Y.

Non ci sono 2 indirizzi di memoria per le operazioni di memoria come "somma i contenuti di LOC1 e LOC2".

## Operazioni dello Stack

Sono disponibili due operazioni "introduci" ed "estrai". Queste trasferiscono A oppure il registro di stato (P) alla sommità dello stack nella memoria aggiornando il puntatore dello stack S. Queste sono PMA e PHP. Le istruzioni inverse sono PLA e PLP (estrai A ed estrai P), che trasferiscono la sommità dello stack rispettivamente in A o P.

## Elaborazione Dati

### *Aritmetica*

Sono disponibili le usuali funzioni di aritmetica in complemento, logica e scorrimento. Le operazioni aritmetiche sono: ADC, SBC. ADC è un'addizione con riporto e perciò non esiste un'addizione senza riporto. Questo è un piccolo svantaggio che richiede un'istruzione CLC prima di qualsiasi addizione. La sottrazione è eseguita da SBC.

È disponibile uno speciale modo decimale che consente l'addizione e sottrazione diretta di numeri espressi in BCD. In molti altri microprocessori è disponibile solo una di queste istruzioni BCD con un codice d'istruzione separato. La presenza del flag decimale moltiplica per due l'effettivo numero di operazioni aritmetiche disponibili.

### *Incremento/Decremento*

Le operazioni di incremento/decremento sono disponibili sulla memoria e sui registri X ed Y ma non sull'accumulatore. Queste sono rispettivamente: INC e DEC, che operano con la memoria, INX, INY e DEX, DEY, che operano con i registri X ed Y.

### *Operazioni Logiche*

Le operazioni logiche sono quelle classiche: AND, ORA, EOR. Verrà chiarito il ruolo di ciascuna di queste istruzioni.

*AND*

Ogni operazione logica è caratterizzata da una tabella della verità che esprime il valore logico del risultato in funzione degli ingressi. La tabella della verità per un AND è la seguente:

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

L'operazione AND è caratterizzata dal fatto che l'uscita è "1" solo se entrambi gli ingressi sono "1". In altre parole se uno degli ingressi è "0" il risultato è sicuramente "0". Questa caratteristica viene impiegata per azzerare una posizione di bit in una parola. Questo è chiamato "mascheratura".

Uno degli impieghi importanti dell'istruzione AND è l'azzeramento o mascheratura di una o più specifiche posizioni di bit in una parola. Si assuma per esempio di voler azzerare le quattro posizioni di bit più a destra di una parola. Questo sarà eseguito dal programma seguente:

```
LDA WORD                WORD CONTENGA '10101010'
AND # % 11110000        '11110000' È LA MASCHERA
```

Si assuma che WORD sia uguale ad '10101010'. Il risultato di questo programma è di lasciare nell'accumulatore il valore '10100000'. "$" viene utilizzato per rappresentare un numero binario.

**Esercizio 4.1:** *Si scriva un programma di due istruzioni che azzeri i bit 1 e 6 di WORD.*

**Esercizio 4.2:** *Cosa succede con la maschera: MASK = '1111111'?*

*ORA*

Quest'istruzione è l'operazione di OR inclusivo. Essa è caratterizzata dalla seguente tabella di verità:

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

L'OR logico è caratterizzato dal fatto che se uno degli operandi è "1", allora il risultato è sempre "1". L'impiego ovvio dell'OR è quello di

porre ad "1" tutti i bit di una parola. Si pongano ad "1" i quattro bit più a destra di WORD. Il programma è:

```
LDA # WORD
ORA # % 00001111
```

Si assuma che WORD contenga '10101010'. Il valore finale dell'accumulatore sarà '10101111".

**Esercizio 4.3:** *Cosa succederebbe se si utilizzasse l'istruzione ORA # % 10101111?*

**Esercizio 4.4:** *Qual'è l'effetto dell'OR con "FF" esadecimale?*

*EOR*

EOR significa "OR-esclusivo". L'OR esclusivo differisce dall'OR inclusivo appena descritto in quanto il risultato è "1" solo se uno degli operandi, e solo uno degli operandi, è uguale ad "1". Se entrambi gli operandi sono uguali ad "1" il normale OR darebbe risultato "1". L'OR esclusivo dà un risultato "0". La tabella della verità è:

0 EOR 0 = 0
0 EOR 1 = 1
1 EOR 0 = 1
1 EOR 1 = 0

L'OR esclusivo è utilizzato per i confronti. Se qualsiasi bit è diverso l'OR esclusivo di due parole sarà diverso da zero. Inoltre nel caso del 6502, l'OR esclusivo è utilizzato *per complementare* una parola poichè non esiste una specifica istruzione di complemento. Questo viene attuato eseguendo l'OR della parola con tutti uni. Il programma è il seguente:

```
LDA # WORD
EOR # % 11111111
```

Si assuma che WORD contenga "10101010". Il valore finale dell'accumultore sarà "01010101". Si può verificare che questo è il complemento del valore originale.

**Esercizio 4.5:** *Qual'è l'effetto di EOR # $ 00?*

**Operazioni di Scorrimento**

Il 6502 standard è equipaggiato con uno scorrimento a sinistra, chiamato ASL (spostamento aritmetico a sinistra) ed uno scorrimento a

destra, chiamato LSR (spostamento logico a destra). Questi saranno descritti in seguito.

Comunque il 6502 ha solo un'istruzione di rotazione a sinistra (ROL)
*Avvertimento:* nessuna versione del 6502 ha un'ulteriore istruzione di rotazione. Si consultino i dati del costruttore per verificare questo fatto. (ROR rotazione a destra).

*Confronti*

I registri X, Y, A possono essere confrontati con la memoria mediante le istruzioni CPX, CPY, CMP.

*Test e Diramazione*

Poichè la verifica è quasi esclusivamente eseguita sui registri dei flag, si esaminino i flag disponibili sul 6502. I contenuti del registro dei flag appaiono nella seguente Figura 4-2.

Si esamini la funzione dei flag procedendo da sinistra a destra.

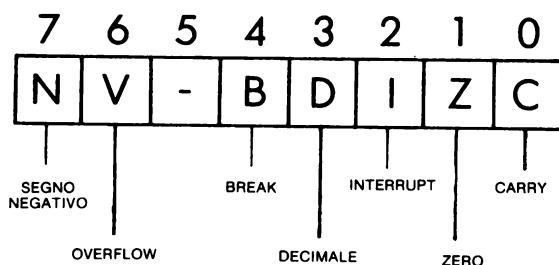



Figura 4.2: Il registro dei flag

*Segno*

Il bit a sinistra è il bit segno, o bit negativo.
Ogni volta che N è 1 indica che il valore di un risultato è negativo nella rappresentazione in complemento a 2. In pratica il flag N è identico al bit 7 di un risultato. Esso è comandato da tutte le istruzioni di trasferimento ed elaborazione dati.

Il flag N è identico al bit 7 dell'accumulatore, nella maggior parte dei casi. Come risultato il bit 7 dell'accumulatore è il solo bit che può essere verificato convenientemente con una singola istruzione. Per verificare

qualsiasi altro bit dell'accumulatore è necessario fare scorrere i suoi contenuti. In tutti i casi dove si vuole verificare velocemente i contenuti di una parola, la posizione di bit preferita sarà perciò il bit 7. Questa è la ragione per cui i bit di stato ingresso/uscita sono normalmente connessi alla posizione 7 del bus dati. Dalla lettura dello stato di un dispositivo I/O si leggerà semplicemente il contenuto del registro di stato esterno nell'accumulatore e quindi il test del bit N.

Il bit successivo all'interno dell'accumulatore che è più facile da verificare è il bit Z (zero). Comunque esso richiede uno scorrimento a destra di 1 nel bit carry così da poter essere verificato. Questo indica se un risultato è zero. Il bit Z non può essere posto al programmatore. Esso è posizionato automaticamente dalle istruzioni.

Le istruzioni che pongono N sono: ADC, AND, ASL, BIT, CMP, CPY, CPX, DEC, DEX, DEY, EOR, INC, INY, LDA, LDX, LDY, LSR, ORA, PLA, PLP, ROL, ROR, TAX, TAY, TXS, TXA, TYA.

*Overflow*

Il ruolo dell'overflow è già stato discusso al Capitolo 3 nel paragrafo sulle operazioni aritmetiche. Esso è utilizzato per indicare che il risultato dell'addizione o sottrazione di numeri in complemento a 2 può essere non corretto a causa di un overflow dal bit 6 al bit 7, cioè nel bit del segno. Una speciale routine di correzione deve essere utilizzata se questo bit vale "1". Se non si utilizza la rappresentazione in complemento a 2, ma il binario diretto, il bit di overflow è equivalente ad un riporto dal bit 6 al bit 7.

Uno speciale impiego di questo bit è determinato dall'istruzione BIT. Un risultato di questa istruzione è di porre il bit "V" identico al bit 6 dei dati da verificare.

Il flag V è condizionato da ADC, BIT, CLV, PLP, RTI, SBC.

*Break*

Questo flag break è posto automaticamente dal processore se un interrupt è causato dal comando BRK. Esso differenzia tra un break programmato ed un interrupt hardware. Nessun' altra istruzione dell'utente lo modificherà.

*Decimale*

L'uso di questo flag è stato già discusso al Capitolo 3 nel paragrafo sui programmi aritmetici. Ogni volta che D è posto ad "1" il processore

opera *nel modo BCD* ed ogni volta che è posto a "0" esso opera in *modo binario.* Questo flag è condizionato da quattro istruzioni: CLD, PLP, RTI, SED.

*Interrupt*

Questo bit della maschera interrupt può essere posto esplicitamente dal programmatore durante il reset oppure durante un interrupt.

Il suo effetto è di inibire qualsiasi ulteriore interrupt.

Le istruzioni che condizionano questo bit sono: BRK, CLI, PLP, RTI, SEI.

*Zero*

Il flag Z indica, quando è uguale ad "1", che il risultato di un trasferimento o di un'operazione è zero. Viene anche influenzato dalle istruzioni di confronto. Non esiste una specifica istruzione che ponga ad 1 od azzeri il bit 0. Comunque lo stesso risultato può essere ottenuto facilmente. Per azzerare il bit carry si può, per esempio, eseguire la seguente istruzione:

LDA # 0

Il bit Z è condizionato da molte istruzioni; ADC, AND, ASL, BIT, CMP, CPY, CPX, DEC, DEX, DEY, EOR, INC, INX, INY, LDA, LDX, LDY, LSR, ORA, PLA, PLP, ROL, ROR, RTI, SBC, TAX, TAY, TXA, TYA.

*Carry*

Si è già visto che il bit carry viene impiegato per un doppio scopo. Il suo primo scopo è di indicare un riporto aritmetico oppure un prestito durante le operazioni aritmetiche. Il suo secondo scopo è di immagazzinare il bit "caduto fuori" da un registro durante le operazioni di scorrimento e rotazione. I due ruoli non devono necessariamente essere confusi e questi non sussistono sui calcolatori più grossi. Comunque questo approccio risparmia tempo nei microprocessori, in particolare per la realizzazione di una moltiplicazione o di una divisione. Il bit carry può essere esplicitamente posto ad 1 od azzerato.

Le istruzioni che condizioneranno il bit carry sono: AD, ASL, CLC, CMP, CPX, CPY, LSR, PLP, ROL, ROR,RTI, SBC, SEC.

Nel 6502 non è possibile verificare se ogni bit del registro dei flag è 1. Ci sono 6 bit e ci sono perciò 12 diverse istruzioni di diramazione. Queste sono:

- BMI (dirama se meno), BPL (dirama se più). Naturalmente queste due istruzioni verificano il bit Z.
- BCC (dirama se carry azzerato) e BCS (dirama se carry posto ad 1): esse provano C.
- BEQ (dirama quando il risultato è zero) e BNE (dirama se risultato non zero). Queste provano Z.
- BVS (dirama quando overflow è posto ad 1) e BVC (dirama se overflow azzerato). Esse provano V.

Queste istruzioni operano la verifica e la diramazione all'interno della stessa istruzione. Tutte le diramazioni specificano uno spostamento relativo all'istruzione corrente. Poichè il campo dello spostamento è 8 bit questo consente uno spostamento da $-128$ a $+127$ (in complemento a due). Lo spostamento è aggiunto all'indirizzo della prima istruzione seguente la diramazione.
Poiché tutte le diramazioni sono lunghe 2 byte questo si risolve in uno spostamento effettivo da $-128+2=-126$ a $+127+2=+129$.
Sono disponibili due ulteriori istruzioni di salto incondizionato: JMP e JSR. JMP è un salto ad un indirizzo a 16 bit, JSR è una chiamata di subroutine. Essa fa saltare ad un nuovo indirizzo e preserva automaticamente il contatore di programma nello stack. Essendo incondizionate queste due istruzioni sono normalmente precedute da un'istruzione di "test e diramazione".
Sono disponibili due istruzioni di ritorno: RTI, ritorno da interrupt, che sarà discusso nel paragrafo degli interrupt, ed RTS, ritorno da subroutine, che estrae un indirizzo di ritorno dallo stack (e lo incrementa).
Sono fornite due istruzioni speciali per la verifica di bit e per i confronti.

L'istruzione BIT esegue un AND tra la locazione di memoria e l'accumulatore. Un aspetto importante è che *essa non cambia i contenuti dell'accumulatore.* Il flag N è posto al valore del bit 7 della locazione di memoria di prova, mentre il flag V uguale al bit 6. Infine il bit Z indica il risultato dell'operazione AND. Z è posto ad "1" se il risultato è "0". Tipicamente una maschera sarà caricata nell'accumulatore ed i succes-

sivi valori di memoria saranno verificati impiegando l'istruzione BIT. Se la maschera contiene un solo "1", per esempio, questo proverà se qualsiasi assegnata parola della memoria contiene un "1" in quella posizione. In pratica questo significa che una maschera potrebbe essere utilizzata solo quando si stanno provando i bit di locazioni di memoria da "0" a "5". Si ricorderà che le locazioni di bit "6" e "7" sono immagazzinate automaticamente rispettivamente nei flag "V" ed "N". Quindi questi non necessitano di essere mascherati.

L'istruzione CMP confronta i contenuti della locazione di memoria con l'accumulatore mediante la sua sottrazione dall'accumulatore stesso. Il risultato del confronto verrà indicato, perciò mediante i bit Z e C. Si può rivelare l'uguaglianza, il maggiore o minore di. Il valore dell'accumulatore non viene cambiato dal confronto. CPX e CPY confronteranno rispettivamente con X e con Y.

**Istruzioni d'Ingresso/Uscita**

Nel 6502 non esistono istruzioni d'ingresso/uscita specializzate.

**Istruzioni di Controllo**

Le istruzioni di controllo comprendono le istruzioni per porre ad 1 ed azzerare i flag. Queste sono: CLC, CLD, CLI, CLV che azzerano rispettivamente i bit C, D, I e V; e SEC, SED, SEI che pongono rispettivamente i bit C, D e I.

L'istruzione BRK è equivalente ad un interrupt software e sarà descritta al Capitolo 7 nel paragrafo degli interrupt.

L'istruzione NOP è un'istruzione che non ha effetti e viene comunemente utilizzata per estendere il timing di un ciclo. Infine due pin speciali del 6502 faranno scattare un meccnismo di interrupt e questo sarà spiegato al Capitolo 6 sulle tecniche d'ingresso/uscita.
Questa è una caratteristica di controllo hardware (pin IRQ ed NMI).

Si esaminerà ora ciascuna istruzione in dettaglio.

Per capire a fondo i vari modi di indirizzamento si incoraggia il lettore ad una prima lettura veloce del paragrafo seguente e ad una lettura più approfondita dopo aver studiato in dettaglio il Capitolo 5 sulle tecniche di indirizzamento.

CAPITOLO 4

# IL SET DI ISTRUZIONI DEL 6502

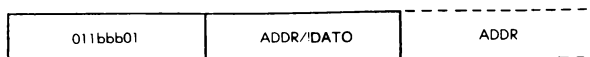
## PARTE II - LE ISTRUZIONI

| | |
|---|---|
| A | Accumulatore |
| M | Indirizzo specificato (memoria) |
| P | Registro di stato |
| S | Puntatore dello Stack |
| X | Registro Indice |
| Y | Registro Indice |
| DATA | Dato specificato |
| HEX | Esadecimale |
| PC | Contatore di Programma |
| PCH | Contatore di Programma alto |
| PCL | Contatore di Programma basso |
| STACK | Contenuti della sommità dello stack |
| V | OR logico |
| Δ | AND logico |
| $\veebar$ | OR esclusivo |
| • | Scambio |
| ← | Riceve il valore (assegnazione) |
| ( ) | Contenuti di |
| (M6) | Posizione di bit 6 all'indirizzo M |

# ADC    Somma con carry

**Funzione:** A ← (A) + DATA + C

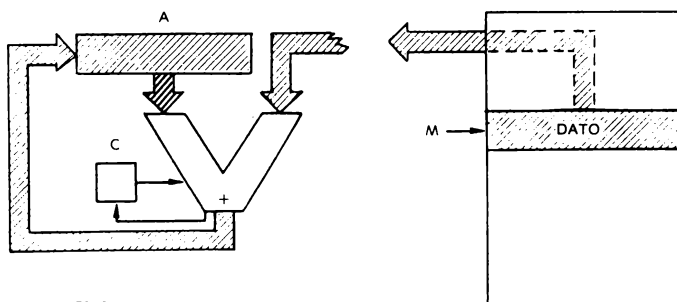
**Formato:**

| 011bbb01 | ADDR/!DATO | ADDR |
|---|---|---|

**Descrizione:**

Somma i contenuti di un indirizzo di memoria o letterale all'accumulatore più il bit carry. Il risultato rimane nell'Accumulatore.

**Note:**

- ADC può operare sia in modo decimale che binario: i flag devono essere posti al valore corretto
- Per sommare senza carry il flag C deve essere azzerato (CLC).

**Percorso dei dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS. X | ABS. Y | (IND. X) | (IND). Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | | 6D | 65 | 69 | 7D | 79 | 61 | 71 | 75 | | | |
| BYTES | | | 3 | 2 | 2 | 3 | 3 | 2 | 2 | 2 | | | |
| CICLI | | | 4 | 3 | 2 | 4* | 4* | 6 | 5* | 4 | | | |
| bbb | | | 011 | 001 | 010 | 111 | 110 | 000 | 100 | 101 | | | |

*: PIU' 1 CICLO SE SI SUPERA LA PAGINA

**Flag:**

| N | V | B | D | I | Z | C |
|---|---|---|---|---|---|---|
| ● | ● | | | | ● | ● |

## Codici di Istruzione:

*: PIU' 1 CICLO SE SI SUPERA LA PAGINA

# AND

## AND Logico

**Funzione:** A ← (A) Δ DATO

**Formato:**

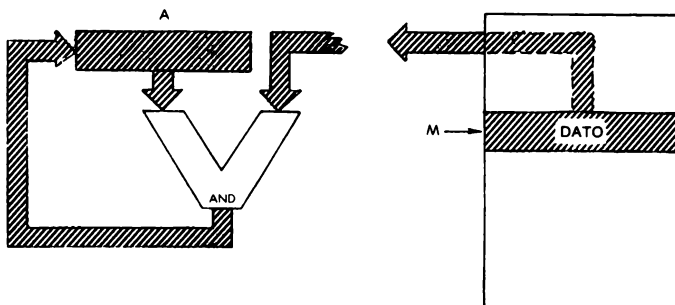
| 001bbb01 | ADDR/DATO | ADDR |
|---|---|---|

**Descrizione:**

Esegue l'AND logico dell'accumulatore e di un dato specifico. Il risultato rimane nell'accumulatore.

La tabella della verità è:

| A\M | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS. X | ABS. Y | (IND. X) | (IND), Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | | 2D | 25 | 29 | 3D | 39 | 21 | 31 | 35 | | | |
| BYTES | | | 3 | 2 | 2 | 3 | 3 | 2 | 2 | 2 | | | |
| CICLI | | | 4 | 3 | 2 | 4* | 4* | 6 | 5* | 4 | | | |
| bbb | | | 011 | 001 | 010 | 111 | 110 | 000 | 100 | 101 | | | |

*: PIU' 1 CICLO SE SI SUPERA LA PAGINA

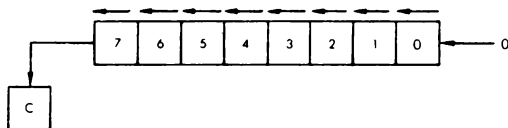
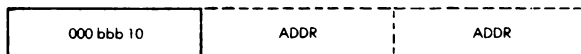
**Flag:**

## Codici di Istruzione:

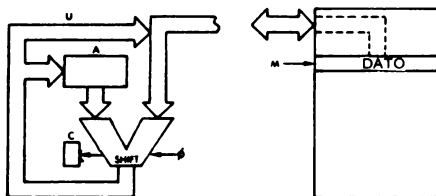
| Modo | Codice | Operando | bbb | HEX | CICLI |
|---|---|---|---|---|---|
| ASSOLUTO | 00101101 | 16 BIT INDIRIZZO | bbb = 011 | HEX = 2D | CICLI = 4 |
| PAGINA-ZERO | 00100101 | ADDR | bbb = 001 | HEX = 25 | CICLI = 3 |
| IMMEDIATO | 00101001 | DATO | bbb = 010 | HEX = 29 | CICLI = 2 |
| ASSOLUTO X | 00111101 | 16 BIT INDIRIZZO | bbb = 111 | HEX = 3D | CICLI = 4* |
| ASSOLUTO Y | 00111001 | 16 BIT INDIRIZZO | bbb = 110 | HEX = 39 | CICLI = 4* |
| (IND). X) | 00100001 | ADDR | bbb = 000 | HEX = 21 | CICLI = 6 |
| (IND). Y | 0011001 | ADDR | bbb = 100 | HEX = 31 | CICLI = 5* |
| PAGINA ZERO X | 00110101 | ADDR | bbb = 101 | HEX = 35 | CICLI = 4 |

*: PIU' 1 CICLO SE SI SUPERA LA PAGINA

**Scorrimento Aritmetico a Sinistra**

**Funzione:**

**Formato:**

**Descrizione:**

Muove i contenuti dell'Accumulatore o di una locazione di memoria a sinistra di una posizione di bit. Da destra entra uno 0. Il bit 7 cade nel carry. Il risultato è depositato nella sorgente cioè nell'accumulatore oppure nella memoria.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS, X | ABS, Y | (IND, X) | (IND), Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | 0A | 0E | 06 | | 1E | | | | 16 | | | |
| BYTES | | 1 | 3 | 2 | | 3 | | | | 2 | | | |
| CICLI | | 2 | 6 | 5 | | 7 | | | | 6 | | | |
| bbb | | 010 | 011 | 001 | | 111 | | | | 101 | | | |

**Flag:**

## Codici di Istruzione:

**Funzione:**

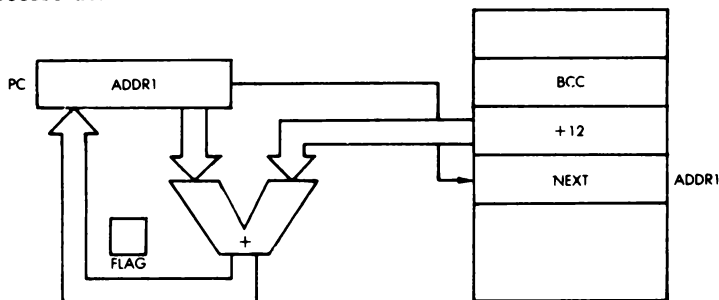
Va ad un indirizzo specificato se C = 0

**Formato:**

| 1001000 | SPOSTAMENTO SUCCESSIVO |
|---|---|

**Descrizione:**

Opera il test del flag carry, opera la diramazione all'indirizzo attuale più lo spostamento assegnato (fino a + 127 o − 128). Se C = 1 non opera. Lo spostamento è sommato all'indirizzo della prima istruzione successiva la BCC. Questo si risolve in uno spostamento effettivo da + 129 a − 126.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = 90, byte = 2, cicli = 2 + 1 se si verifica la diramazione
+ 2 se si passa ad un'altra pagina

**Flag:**

# BCS

**Opera Diramazione se Carry è posto ad 1**

**Funzione:**

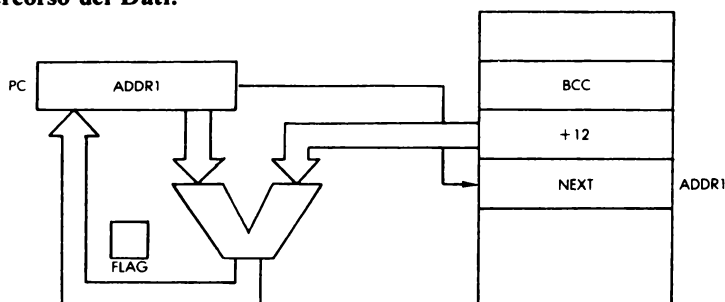
Va all'indirizzo specificato se C = 1

**Formato:**

| 10110000 | SPOSTAMENTO SUCCESSIVO |
|---|---|

**Descrizione:**

Opera il test del flag carry. Se C = 1 opera la diramazione all'indirizzo attuale più lo spostamento assegnato (fino a + 127 o − 128). Se C = 0 non opera. Lo spostamento è sommato all'indirizzo della prima istruzione successiva a BCC. Questo si risolve in uno spostamento effettivo da + 129 a − 126.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto relativo:
HEX = B0, byte = 2, cicli = 2

+ 1 se si verifica la diramazione
+ 2 se si passa ad un'altra pagina.

**Flag:**

# BEQ Opera Diramazione Uguale a Zero

**Funzione:**

Va ad un indirizzo specificato se Z = 1 (risultato = 0)

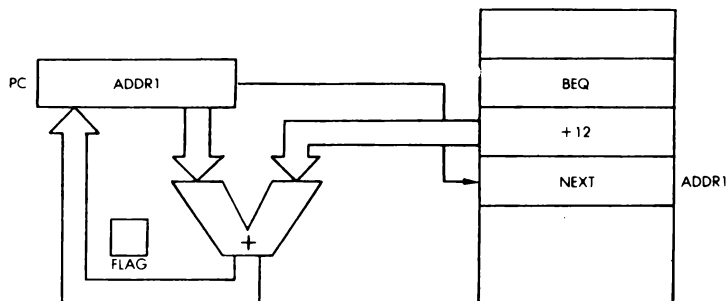
**Formato:**

| 11110000 | SPOSTAMENTO SUCCESSIVO |
|---|---|

**Descrizione:**

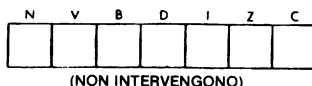
Opera il test del flag Z. Se Z = 1 opera la diramazione all'indirizzo attuale più lo spostamento assegnato (fino a + 127 o − 128). Se Z = 0 non opera.

Lo spostamento è sommato all'indirizzo della prima istruzione successiva la BEQ. Questo si risolve in un effettivo spostamento da + 129 a − 126.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto relativo

HEX = F0, byte = 2, cicli = 2 + 1 se si verifica la diramazione
+ 2 se passa ad un'altra pagina

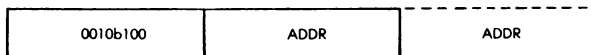
**Flag:**

# BIT

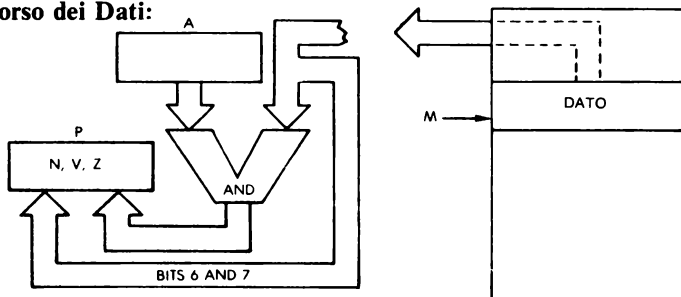
**Confronta i bit di memoria con l'accumulatore**

**Funzione:**

$Z \leftarrow \overline{(A) \wedge (M)},\ N \leftarrow (M^7),\ V \leftarrow (M^6)$

**Formato:**

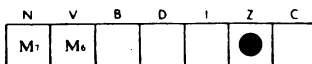
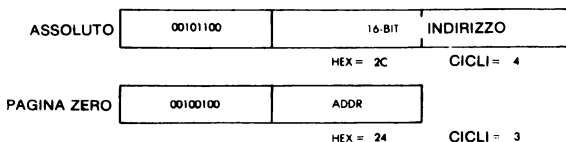
| 0010b100 | ADDR | ADDR |
|---|---|---|

**Descrizione:**

Viene eseguito ma non immagazzinato l'AND di A ed M. Il risultato del confronto è indicato da Z. Z = 1 se il confronto è soddisfatto, altrimenti è Z = 0. Inoltre i bit 6 e 7 del dato di memoria sono trasferiti nei flag V ed N del registro di stato.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS, X | ABS, Y | (IND, X) | (IND), Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | | 2C | 24 | | | | | | | | | |
| BYTES | | | 3 | 2 | | | | | | | | | |
| CICLI | | | 4 | 3 | | | | | | | | | |
| bbb | | | 011 | 001 | | | | | | | | | |

**Flag:**

| N | V | B | D | I | Z | C |
|---|---|---|---|---|---|---|
| $M_7$ | $M_6$ | | | | ● | |

**Codici di Istruzione:**

# BMI Opera Diramazione se Negativo

**Funzione:**

Va ad un indirizzo specificato se N = 1 (risultato < 0)

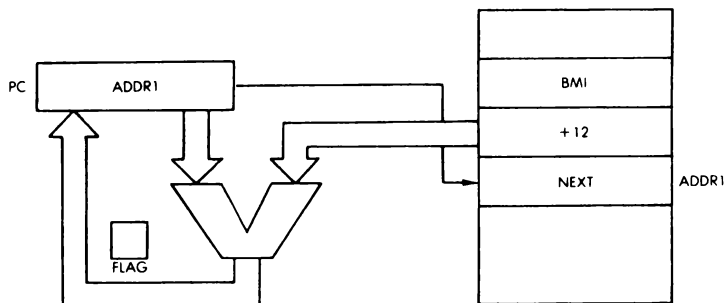
**Formato:**

| 00110000 | SPOSTAMENTO SUCCESSIVO |
|---|---|

**Descrizione:**

Opera il test del flag N (segno). Se N = 1 opera la diramazione all'indirizzo attuale più lo spostameno assegnato (fino a + 127 o — 128). Se N = 0 non opera.

Lo spostamento è sommato all'indirizzo della prima istruzione successiva BEQ. Questo si risolve in uno spostamento effettivo da + 129 a − 126.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto relativo:
HEX = 30, byte = 2, cicli = 2 + 1 se si verifica la diramazione
+ 2 se si passa ad un'altra pagina

**Flag:**

# BNE Opera Diramazione se non uguale a zero

**Funzione:**

Va all'indirizzo specificato se Z = 0 (risultato = 0)

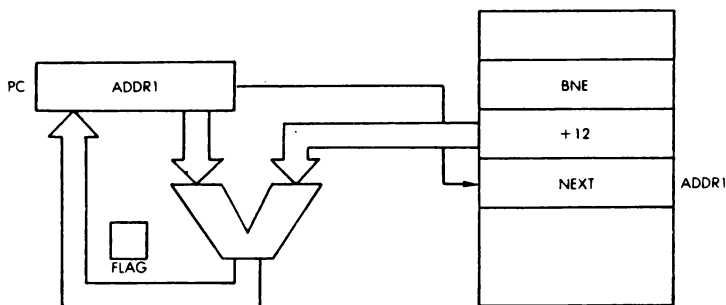
**Formato:**

| 11010000 | SPOSTAMENTO SUCCESSIVO |
|---|---|

**Descrizione:**

Verifica il risultato (flag Z). Se il risultato non è uguale a zero (Z = 0), opera la diramazione all'indirizzo attuale più lo spostamento assegnato (fino a + 127 o – 128). Se N = 0 non opera.

Lo spostamento è sommato all'indirizzo della prima istruzione successiva la BEQ. Questo si risolve in uno spostamento effettivo da + 129 a – 126.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto relativo:
HEX = D0, byte = 2, cicli = 2

+ 1 se si verifica la diramazione
+ 2 se si passa ad un'altra pagina

**Flag:**

## BPL Opera Diramazione se positivo

**Funzione:**

Va ad un indirizzo specificato se N = 0 (risultato $\geqslant 0$)

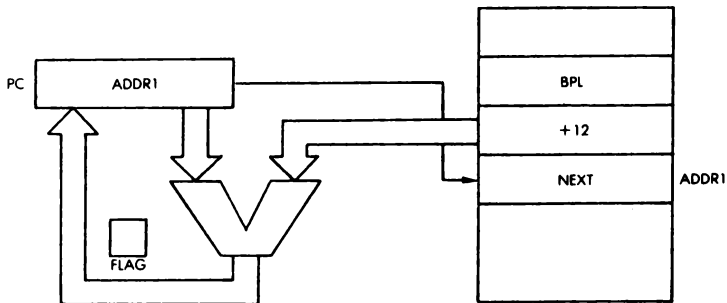
**Formato:**

| 00010000 | SPOSTAMENTO SUCCESSIVO |
|---|---|

**Descrizione:**

Opera il test del flag N (segno). Se N = 0 (risultato positivo) opera la diramazione all'indirizzo attuale più lo spostameno assegnato (fino a + 127 o − 128). Se N = 1 non opera.

Lo spostamento è sommato all'indirizzo della prima istruzione successiva BEQ. Questo si risolve in uno spostamento effettivo da + 129 a − 126.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto relativo:
HEX = 10, byte = 2, cicli = 2 + 1 se si verifica la diramazione
+ 2 se si passa ad un'altra pagina

**Flag:**

# BRK **Break**

**Funzione:**

STACK (PC) + 2, STACK (P), PC ← (FFFE, FFFF)

**Formato:**

| 00000000 |
|---|

**Descrizione:**

Opera come interrupt: il contatore di programma è introdotto nello stack e quindi il registro di stato P. I contenuti delle locazioni di memoria FFFE ed FFFF sono quindi depositati rispettivamente in PCL e PCH. Il valore di P immagazzinato nello stack ha il flag B posto ad 1 per differenziare BRK da IRQ.

Importante: diversamente da un interrupt, PC + 2 è conservato. Questa può non essere l'istruzione successiva e si può rendere necessaria una correzione. Questo è dovuto all'impiego normale di BRK per aggiustare i programmi esistenti dove BRK sostituisce un'istruzione di 2 byte.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = 00, byte = 2, cicli = 7

**Flag:**

(NOTA: B È POSTO NELLO STACK)

# BVC Opera Diramazione se Overflow è zero

**Funzione:**

Va all'indirizzo specificato se V = 0

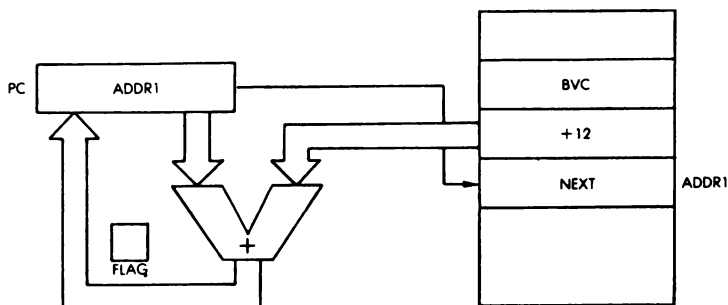
**Formato:**

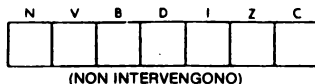
| 0101000 | SPOSTAMENTO SUCCESSIVO |
|---|---|

**Descrizione:**

Verifica il flag overflow (V). Se non c'è overflow (V = 0) opera la diramazione all'indirizzo attuale più lo spostamento assegnato (fino a + 127 o — 128). Se V = 1 non opera.

Lo spostamento è sommato all'indirizzo della prima istruzione successiva la BEQ. Questo si risolve in uno spostamento effettivo da + 129 a — 126.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto relativo:
HEX = 50, byte = 2, cicli = 2 + 1 se si verifica la diramazione
+ 2 se si passa ad un'altra pagina

**Flag:**

# BVS Opera Diramazione se overflow è posto ad 1

**Funzione:**

Va all'indirizzo specificato se V = 1.

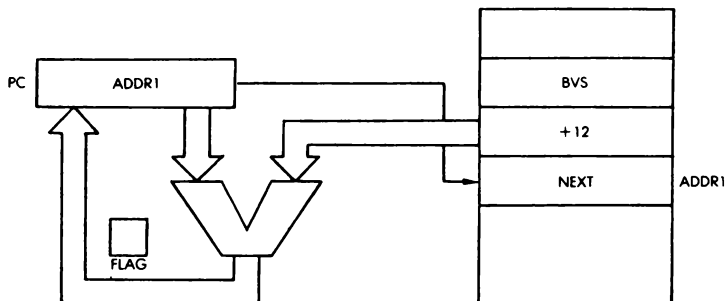
**Formato:**

| 01110000 | SPOSTAMENTO SUCCESSIVO |
|---|---|

**Descrizione:**

Verifica il flag overflow (V). Se si è verificato un overflow (V = 1), opera un diramazione all'indirizzo attuale più lo spostamento assegnato (fino a + 127 o — 128). Se V = 0 non opera.

Lo spostamento è sommato all'indirizzo della prima istruzione successiva BVS. Questo si risolve in uno spostamento effettivo da + 129 a − 126.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto relativo:
HEX = 70, byte = 2, cicli = 2

+ 1 se si verifica la diramazione
+ 2 se si passa ad un'altra pagina

**Flag:**

# CLC

**Azzera carry**

**Funzione:**

C ← Ø

**Formato:** 00011000

**Descrizione:**

Viene azzerato il bit carry. Questo è spesso necessario prima di una ADC.

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = 18, byte = 1, cicli = 2

**Flag:**

# CLD

**Azzera il flag decimale**

**Funzione:**

D ← Ø

**Formato:**

| 11011000 |
|---|

**Descrizione:**

Viene azzerato il flag D preselezionando così il modo binario per ADC ed SBC.

**Modi di Indirizzamento:**

Soltanto implicato:
HEX ≓ D8, byte = 1, cicli = 2

**Flag:**

## CLI — Azzera la maschera di interrupt

**Funzione:**

I ← Ø

**Formato:**

| 01011000 |
|---|

**Descrizione:**

Il bit della maschera interrupt viene posto a 0. Questo abilita gli interrupt. Una routine di manipolazione degli interrupt deve sempre azzerare il bit I, diversamente agli altri interrupt possono andare persi.

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = 58, byte = 1, cicli = 2

**Flag:**

# CLV — Azzera il flag di overflow

**Funzione:**

V ← ∅

**Formato:** 10111000

**Descrizione:**

Viene azzerato il flag di overflow

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = B8, byte = 1, cicli = 2

**Flag:**

# CMP  Confronta con l'accumulatore

**Funzione:**

(A) — DATO → NZC:

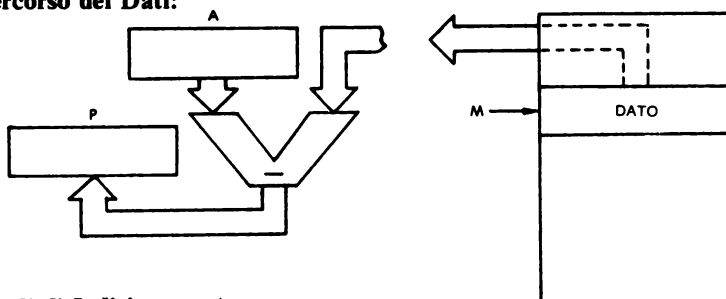
| + (A > DATO) | = | − (A < DATO) |
|---|---|---|
| −01 | 011 | −00 |

**Formato:**

| 110bbb01 | ADDR/DATO | ADDR |
|---|---|---|

**Descrizione:**

I contenuti specificati vengono sottratti da A. Il risultato non è immagazzinato ma vengono condizionati i flag NZC in dipendenza se il risultato è positivo, nullo o negativo. Il valore dell'accumulatore non viene cambiato. CMP è normalmente seguito da una diramazione: BCC rivela A < DATO, BEQ rivela A = DATO e BEQ seguito da BCS rivela A ≥ DATO.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS, X | ABS, Y | (IND, X) | (IND), Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | | CD | C5 | C9 | DD | D9 | C1 | D1 | D5 | | | |
| BYTES | | | 3 | 2 | 2 | 3 | 3 | 2 | 2 | 2 | | | |
| CICLI | | | 4 | 3 | 2 | 4* | 4* | 6 | 5* | 4 | | | |
| bbb | | | 011 | 001 | 010 | 111 | 110 | 000 | 100 | 101 | | | |

*: PIU' 1 CICLO SE SI SUPERA LA PAGINA

**Flag:**

## Codici di Istruzione:

*: PIU' 1 CICLO SE SI SUPERA LA PAGINA

# CPX Confronta col registro X

**Funzione:**

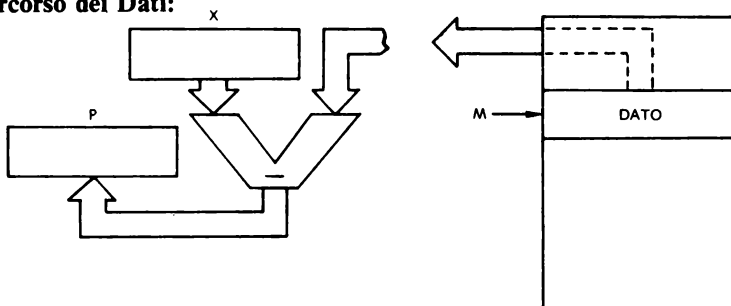
X — DATO → NZC:

| + (X > DATO) | = | − (X <DATO) |
|---|---|---|
| −01 | 011 | −00 |

**Formato:**

| 111ØbbØØ | ADDR/DATO | ADDR |
|---|---|---|

**Descrizione:**

I contenuti specificati sono sottratti da X. Il risultato non viene immagazzinato ma vengono condizionati i flag NCZ in dipendenza se il risultato è positivo, negativo o nullo. Il valore dell'accumulatore non viene cambiato. CPX è normalmente seguito da una diramazione: BCC rivela (X) < DATO, BEQ rivela (X) < DATO e BEQ seguito da BCS rivela (X) > DATO. BCS rivela X≥ DATO.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS.X | ABS.Y | (IND.X) | (IND).Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | | EC | E4 | E0 | | | | | | | | |
| BYTES | | | 3 | 2 | 2 | | | | | | | | |
| CICLI | | | 4 | 3 | 2 | | | | | | | | |
| bb | | | 11 | 01 | 00 | | | | | | | | |

**Flag:**

| N | V | B | D | I | Z | C |
|---|---|---|---|---|---|---|
| ● | | | | | ● | ● |

## Codici di Istruzione:

| | | | |
|---|---|---|---|
| ASSOLUTO | 11101100 | 16-BIT INDIRIZZI | |
| | bb = 11 | HEX = EC | CICLI = 4 |
| PAGINA ZERO | 11100100 | ADDR | |
| | bb = 01 | HEX = E4 | CICLI = 3 |
| IMMEDIATO | 11100000 | DATO | |
| | bb = 00 | HEX = EO | CICLI = 2 |

# CPY

## Confronta col registro Y

**Funzione:**

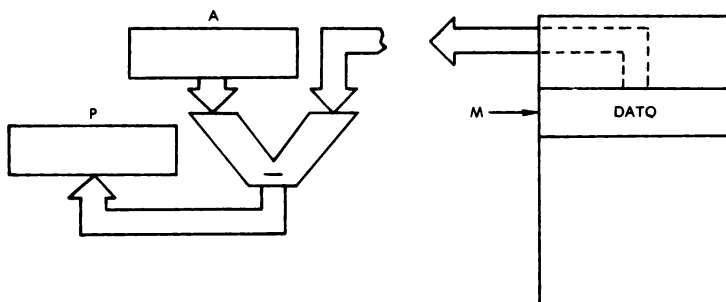
(Y) — DATO → NZC:

| +(Y > DATO) | = | −(Y < DATO) |
|---|---|---|
| −01 | 011 | −00 |

**Formato:**

| 1100bb00 | ADDR/DATO | ADDR |
|---|---|---|

**Descrizione:**

I contenuti specificati sono sottratti da Y. Il risultato non è immagazzinato ma i flag NCZ sono condizionati in dipendenza se il risultato è positivo, nullo o negativo. Il valore dell'accumulatore non viene cambiato. CPY è normalmente seguito da una diramazione: BCC rivela (Y) < DATO, BEQ rivela (Y) < DATO e BEQ seguito da BCS rivela (Y) > DATO. BCS rivela X≥ DATO.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS. X | ABS. Y | (IND, X) | (IND), Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | | CC | C4 | C0 | | | | | | | | |
| BYTES | | | 3 | 2 | 2 | | | | | | | | |
| CICLI | | | 4 | 3 | 2 | | | | | | | | |
| bb | | | 11 | 01 | 00 | | | | | | | | |

**Flag:**

| N | V | B | D | I | Z | C |
|---|---|---|---|---|---|---|
| ● | | | | | ● | ● |

## Codici di Istruzione:

| | | | |
|---|---|---|---|
| ASSOLUTO | 11001100 | 16-BIT | INDIRIZZO |
| | bb = 11 | HEX = CC | CICLI = 4 |
| PAGINA-ZERO | 11000100 | ADDR | |
| | bb = 01 | HEX = C4 | CICLI = 3 |
| IMMEDIATO | 11000000 | DATO | |
| | bb = 00 | HEX = CO | CICLI = 2 |

# DEC — Decrementa

**Funzione:**

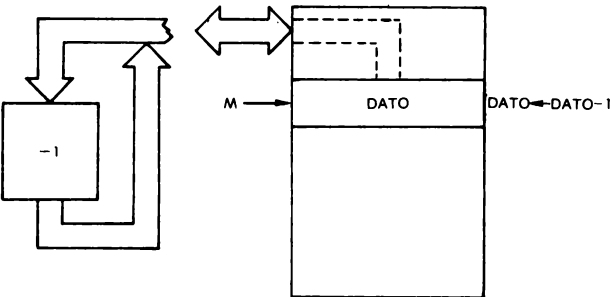
M ← (M) — 1

**Formato:**

| 110bb110 | ADDR | ADDR |
|---|---|---|

**Descrizione:**

I contenuti dell'indirizzo di memoria specificato sono decrementati di 1. Il risultato è ri-immagazzinato all'indirizzo di memoria specificato.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS. X | ABS. Y | (IND. X) | (IND), Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | | CE | C6 | | DE | | | | D6 | | | |
| BYTES | | | 3 | 2 | | 3 | | | | 2 | | | |
| CICLI | | | 6 | 5 | | 7 | | | | 6 | | | |
| bb | | | 01 | 00 | | 11 | | | | 10 | | | |

**Flag:**

| N | V | B | D | I | Z | C |
|---|---|---|---|---|---|---|
| ● | | | | | ● | |

## Codici di Istruzione:

# DEX

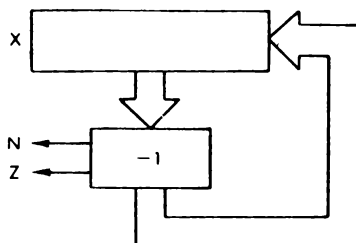
**Decrementa X**

**Funzione:**

X ← (X) — 1

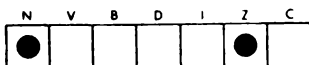
**Formato:** 11001010

**Descrizione:**

I contenuti di X vengono decrementati di 1. Consente l'utilizzazione di X come contatore.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = CA, byte = 1, cicli = 2

**Flag:**

# DEY — Decrementa Y

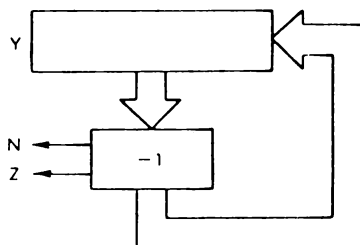
**Funzione:**

$Y \leftarrow (Y) - 1$

**Formato:** 10001000

**Descrizione:**

I contenuti di Y vengono decrementati di 1. Consente l'utilizzazione di Y come contatore.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = 88, byte = 1, cicli = 2

**Flag:**

# EOR — Or esclusivo con l'accumulatore

**Funzione:**

A ← (A) V DATO

**Formato:**

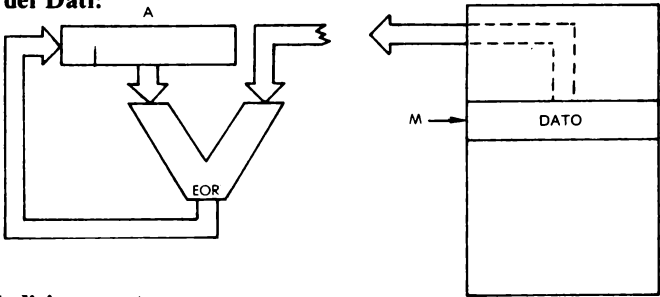
| 010bbb01 | ADDR/DATO | ADDR |
|---|---|---|

**Descrizione:**

Viene operato l'or esclusivo dei contenuti dell'accumulatore con il dato specificato. La tabella della verità è:

| | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

Note: l'EOR con "— 1" può essere utilizzato per complementare.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS, X | ABS, Y | (IND, X) | (IND), Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | | 4D | 45 | 49 | 5D | 59 | 41 | 51 | 55 | | | |
| BYTES | | | 3 | 2 | 2 | 3 | 3 | 2 | 2 | 2 | | | |
| CICLI | | | 4 | 3 | 2 | 4* | 4* | 6 | 5* | 4 | | | |
| bbb | | | 011 | 001 | 010 | 111 | 110 | 000 | 100 | 101 | | | |

*: PIU' 1 CICLO SE SI SUPERA LA PAGINA

**Flag:**

| N | V | B | D | I | Z | C |
|---|---|---|---|---|---|---|
| ● | | | | | ● | |

## Codici di Istruzione:

*: PIU' 1 CICLO SE SI SUPERA LA PAGINA

# INC **Incrementa la memoria**

**Funzione:**

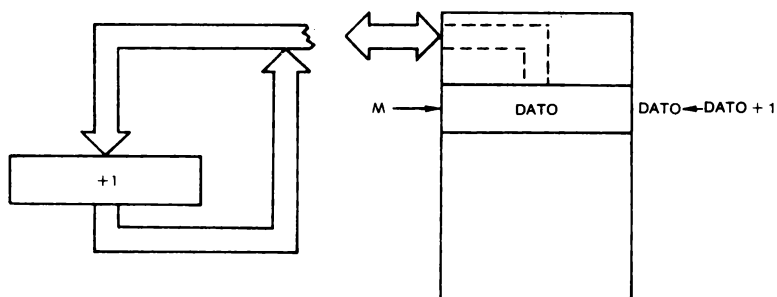
$M \leftarrow (M) + 1$

**Formato:**

| 111bb110 | ADDR | ADDR |
|---|---|---|

**Descrizione:**

I contenuti della locazione di memoria specificata sono incrementati di uno e quindi riposizionati nella locazione stessa.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS. X | ABS. Y | (IND. X) | (IND. Y) | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | | EE | E6 | | FE | | | | F6 | | | |
| BYTES | | | 3 | 2 | | 3 | | | | 2 | | | |
| CICLI | | | 6 | 5 | | 7 | | | | 6 | | | |
| bb | | | 01 | 00 | | 11 | | | | 10 | | | |

**Flag:**

| N | V | B | D | I | Z | C |
|---|---|---|---|---|---|---|
| ● | | | | | ● | |

## Codici di Istruzione:

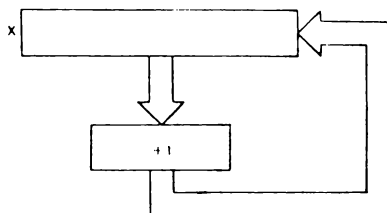
# INX    Incrementa X

**Funzione:**

$X \leftarrow (X) + 1$

**Formato:** 11101000

**Descrizione:**

I contenuti di X sono incrementati di uno. Questo consente l'impiego di X come contatore.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = E8, byte = 1, cicli = 2

**Flag:**

# INY **Incrementa Y**

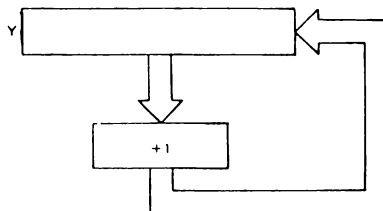
**Funzione:**

Y ← (Y) + 1

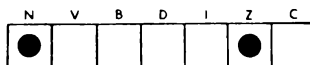
**Formato:** 11001000

**Descrizione:**

I contenuti di Y sono incrementati di uno. Questo consente l'impiego di Y come contatore.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

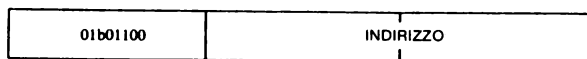
Soltanto implicato:
HEX = C8, byte = 1, cicli = 2

**Flag:**

# JMP                Salta all'indirizzo

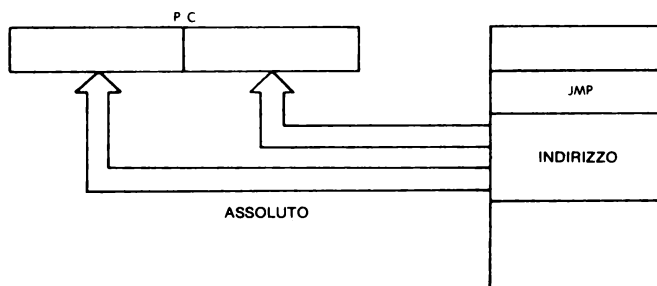
**Funzione:**

PC ← INDIRIZZO

**Formato:**

| 01b01100 | INDIRIZZO |
|---|---|

**Descrizione:**

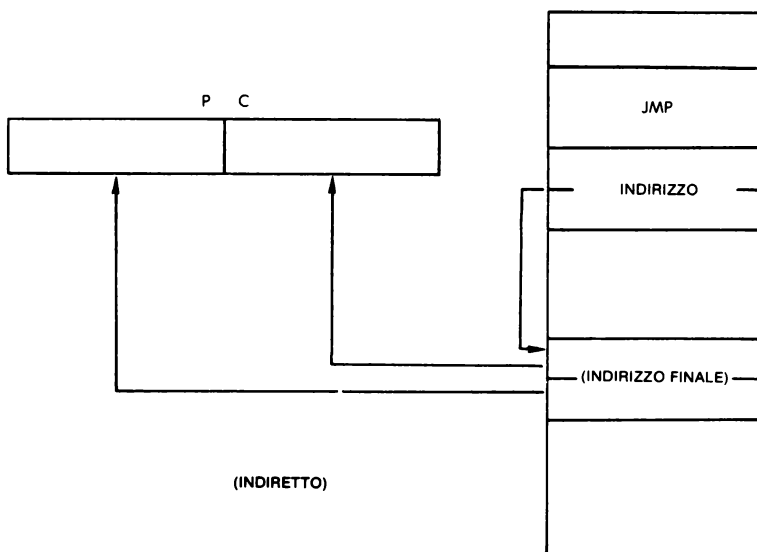
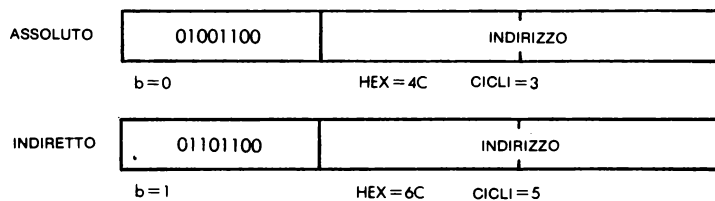
Viene caricato un nuovo indirizzo nel contatore di programma originando un salto rispetto all'esecuzione sequenziale del programma. L'indirizzo può essere specificato in modo assoluto oppure indiretto.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS, X | ABS, Y | (IND, X) | (IND), Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | | 4C | | | | | | | | | | 6C |
| BYTES | | | 3 | | | | | | | | | | 3 |
| CICLI | | | 3 | | | | | | | | | | 5 |
| b | | | 0 | | | | | | | | | | 1 |

**Flag:**

## Codici di Istruzione:

# JSR

## Salta alla subroutine

**Funzione:**

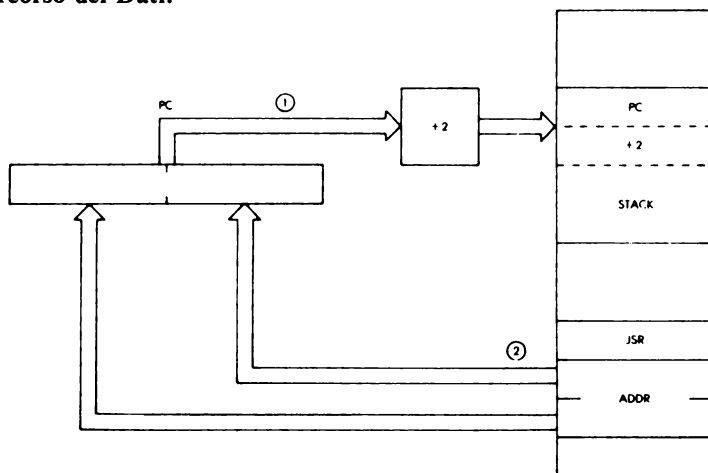
STACK ← (PC) + 2
PC ← INDIRIZZO

**Formato:**

| 00100000 | INDIRIZZO |
|---|---|

**Descrizione:**

I contenuti del contatore di programma + 2 sono conservati nello stack. (Questo è l'indirizzo dell'istruzione successiva la JSR). L'indirizzo della subroutine è quindi caricato nel PC. Quest'operazione è anche detta "chiamata della subroutine".

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto assoluto:
HEX = 20, byte = 3, cicli = 6

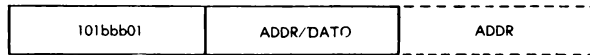
**Flag:**

# LDA Carica l'Accumulatore

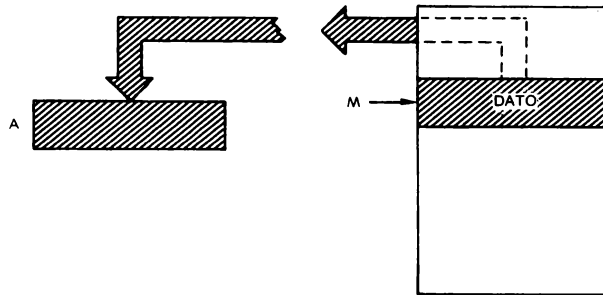
**Funzione:**

A ← DATO

**Formato:**

| 101bbb01 | ADDR/DATO | ADDR |
|---|---|---|

**Descrizione:**

L'accumulatore è caricato con un nuovo dato.

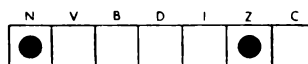
**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA 0 | IMMEDIATO | ABS, X | ABS, Y | (IND, X) | (IND), Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | | AD | A5 | A9 | BD | B9 | A1 | B1 | B5 | | | |
| BYTES | | | 3 | 2 | 2 | 3 | 3 | 2 | 2 | 2 | | | |
| CICLI | | | 4 | 3 | 2 | 4* | 4* | 6 | 5* | 4 | | | |
| bbb | | | 011 | 001 | 010 | 111 | 110 | 000 | 100 | 101 | | | |

*: PIU' 1 CICLO SE SI SUPERA LA PAGINA

**Flag:**

| N | V | B | D | I | Z | C |
|---|---|---|---|---|---|---|
| ● | | | | | ● | |

## Codici di Istruzione:

| Modo | Codice | Operando | bbb | HEX | CICLI |
|---|---|---|---|---|---|
| ASSOLUTO | 10101101 | 16 BIT INDIRIZZO | bbb = 011 | HEX = AD | CICLI = 4 |
| PAGINA ZERO | 10100101 | ADDR | bbb = 001 | HEX = A5 | CICLI = 3 |
| IMMEDIATO | 10101001 | DATO | bbb = 010 | HEX = A9 | CICLI = 2 |
| ASSOLUTO X | 10111101 | 16 BIT INDIRIZZO | bbb = 111 | HEX = BD | CICLI = 4* |
| ASSOLUTO Y | 10111001 | 16 BIT INDIRIZZO | bbb = 110 | HEX = B9 | CICLI = 4* |
| (IND), X | 10100001 | ADDR | bbb = 000 | HEX = A1 | CICLI = 6 |
| (IND), Y | 10110001 | ADDR | bbb = 100 | HEX = B1 | CICLI = 5* |
| PAGINA ZERO X | 10110101 | ADDR | bbb = 101 | HEX = B5 | CICLI = 4 |

*: PIU' 1 CICLO SE SI SUPERA LA PAGINA

# LDX

**Carica il registro X**

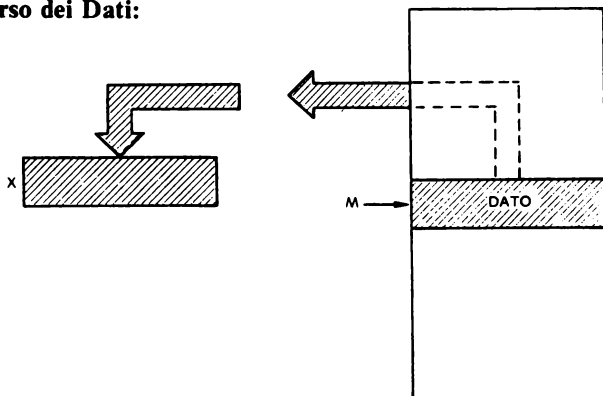
**Funzione:**

X ← DATO

**Formato:**

| 101bbb10 | ADDR/DATO | ADDR |
|---|---|---|

**Descrizione:**

Il registro indice X viene caricato con un dato proveniente dall'indirizzo specificato.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS. X | ABS. Y | (IND. X) | (IND), Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | | AE | A6 | A2 | | BE | | | | B6 | | |
| BYTES | | | 3 | 2 | 2 | | 3 | | | | 2 | | |
| CICLI | | | 4 | 3 | 2 | | 4* | | | | 4 | | |
| bbb | | | 011 | 001 | 000 | | 111 | | | | 110 | | |

*: PIU' 1 CICLO SE SI SUPERA LA PAGINA

**Flag:**

## Codici di Istruzione:

*: PIU' 1 CICLO SE SI SUPERA LA PAGINA

# LDY    Carica il registro Y

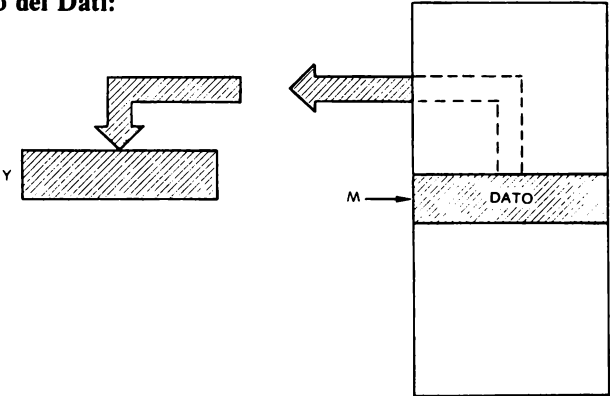
**Funzione:**

Y ← DATO

**Formato:**

| 101bbb00 | ADDR/DATO | ADDR |
|---|---|---|

**Descrizione:**

Il registro indice Y viene caricato con un dato proveniente dall'indirizzo specificato.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS. X | ABS. Y | (IND. X) | (IND). Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | | AC | A4 | A0 | BC | | | | B4 | | | |
| BYTES | | | 3 | 2 | 2 | 3 | | | | 4 | | | |
| CICLI | | | 4 | 3 | 2 | 4* | | | | 4 | | | |
| bbb | | | 011 | 001 | 000 | 111 | | | | 101 | | | |

*: PIU' 1 CICLO SE SI SUPERA LA PAGINA

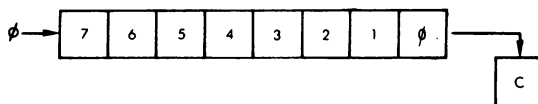
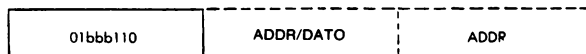
**Flag:**

## Codici di Istruzione:

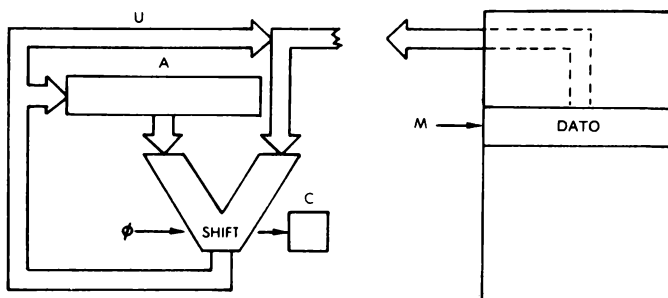
*: PIU' 1 CICLO SE SI SUPERA LA PAGINA

# LSR

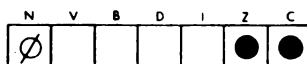
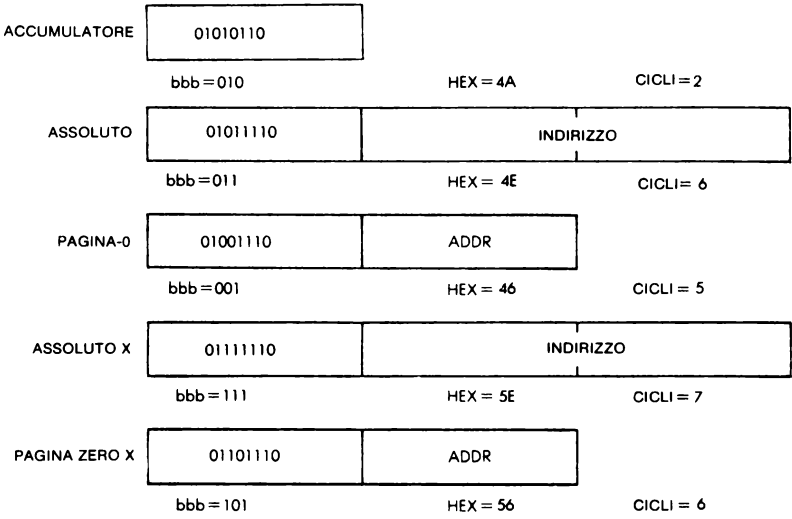
## Scorrimento logico a destra

**Funzione:**

**Formato:**

**Descrizione:**

Fa scorrere a destra di una posizione di bit i contenuti specificati (accumulatore o memoria). Uno "0" è forzato nel bit 7. Il bit 0 è trasferito nel carry. Il dato che ha subito lo scorrimento è depositato nella sorgente cioè nell'accumulatore o nella memoria.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS. X | ABS. Y | (IND. X) | (IND), Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | 4A | 4E | 46 | | 5E | | | | 56 | | | |
| BYTES | | 1 | 3 | 2 | | 3 | | | | 2 | | | |
| CICLI | | 2 | 6 | 5 | | 7 | | | | 6 | | | |
| bbb | | 010 | 011 | 001 | | 111 | | | | 101 | | | |

**Flag:**

## Codici di Istruzione:

# NOP **Nessuna operazione**

**Funzione:**

Nessuna

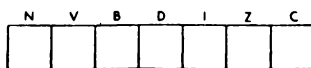
**Formato:** 11101010

**Descrizione:**

Non opera per 2 cicli. Può essere utilizzata per temporizzare un ciclo di ritardo o per riempire un programma.

**Modi di Indirizzamento:**

Soltanto implicato
HEX = EA, byte = 1, cicli = 2

**Flag:**

# ORA OR inclusivo con l'accumulatore

**Funzione:**

A ← (A) V DATO

**Formato:**

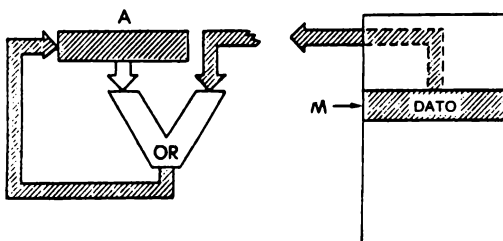
| 000bbb01 | ADDR/DATO |
|---|---|

**Descrizione:**

Esegue l'OR inclusivo logico di A con un dato specificato. Il risultato è immagazzinato in A. Può essere utilizzato per forzare un "1" in una locazione di bit selezionata.

Tabella della verità:

| | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS. X | ABS. Y | (IND. X) | (IND). Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | | 0D | 05 | 09 | 1D | 19 | 01 | 11 | 15 | | | |
| BYTES | | | 3 | 2 | 2 | 3 | 3 | 2 | 2 | 2 | | | |
| CICLI | | | 4 | 3 | 2 | 4* | 4* | 6 | 5* | 4 | | | |
| bbb | | | 011 | 001 | 010 | 111 | 110 | 000 | 100 | 101 | | | |

*: PIU' 1 CICLO SE SI SUPERA LA PAGINA

**Flag:**

| N | V | B | D | I | Z | C |
|---|---|---|---|---|---|---|
| ● | | | | | ● | |

## Codici di Istruzione:

| Modo | Codice | Operando | bbb | HEX | CICLI |
|---|---|---|---|---|---|
| ASSOLUTO | 00001101 | 16-BIT INDIRIZZO | 011 | 0D | 4 |
| PAGINA ZERO | 00000101 | ADDR | 001 | 05 | 3 |
| IMMEDIATO | 00001001 | DATO | 010 | 09 | 2 |
| ASSOLUTO X | 00011101 | 16-BIT INDIRIZZO | 111 | 1D | 4* |
| ASSOLUTO Y | 00011001 | 16-BIT INDIRIZZO | 110 | 19 | 4* |
| (IND, X) | 00000001 | ADDR | 000 | 01 | 6 |
| (IND), Y | 00010001 | ADDR | 100 | 11 | 5* |
| PAGINA 0 X | 00010101 | ADDR | 101 | 15 | 4 |

*: PIU' 1 CICLO SE SI SUPERA LA PAGINA

# PHA Spingi A

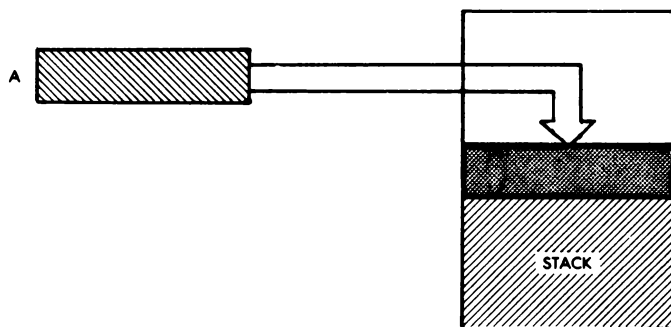
**Funzione:**

STACK ← (A)
S ← (S) — 1

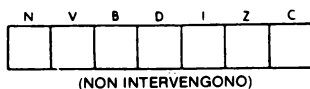
**Formato:** 01001000

**Descrizione:**

I contenuti dell'accumulatore vengono spinti nello stack. Il puntatore dello stack viene aggiornato. A è invariato.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = 48, byte = 1, cicli = 3

**Flag:**

# PHP

**Spingi lo stato del processore**

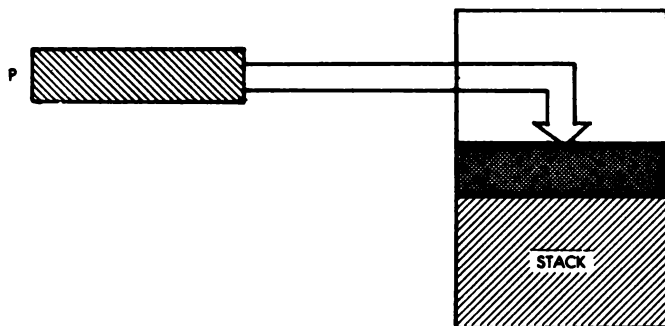
**Funzione:**

STACK ← (P)
S ← (S) — 1

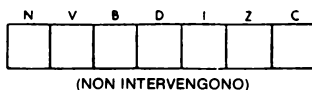
**Formato:** 00001000

**Descrizione:**

I contenuti del registro di stato P sono spinti nello stack. Il puntatore dello stack viene aggiornato. A è invariato.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = 08, byte = 1, cicli = 3

**Flag:**

# PLA **Estrai l'Accumulatore**

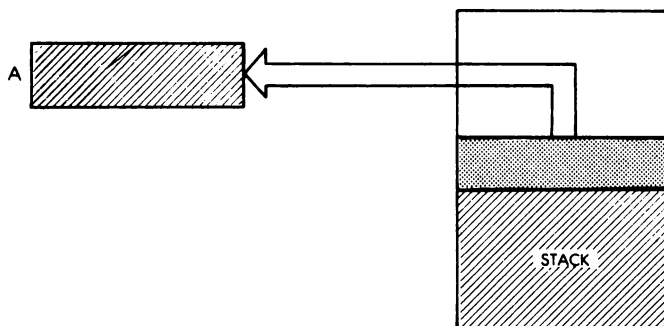
**Funzione:**

A ← (STACK)
S ← (S) + 1

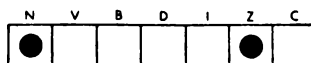
**Formato:** 01101000

**Descrizione:**

Estrae la parola alla sommità dello stack depositandola nell'accumulatore. Incrementa il puntatore dello stack.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = 68, byte = 1, cicli = 4

**Flag:**

# PLP Estrai lo stato del Processore dallo stack

**Funzione:**

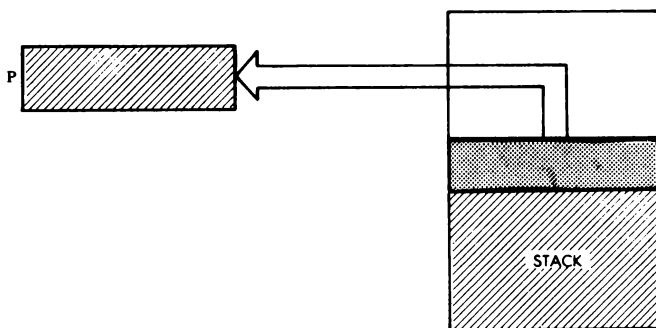
P ← (STACK)
S ← (S) + 1

**Formato:**

| 00101000 |
|---|

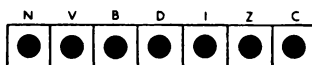
**Descrizione:**

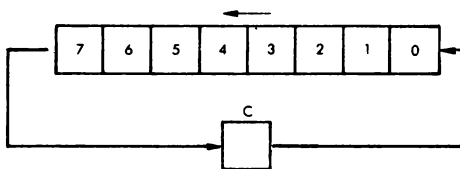
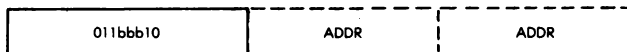
La parola alla sommità dello stack viene estratta e trasferita nel registro di stato P. Il puntatore dello stack è incrementato.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

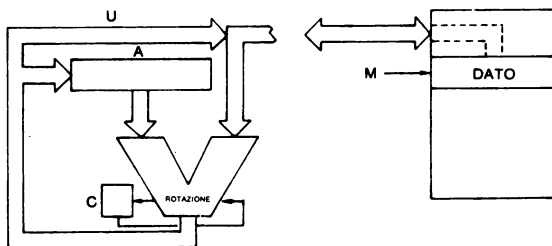
Soltanto implicato:
HEX = 28, byte = 1, cicli = 4

**Flag:**

# ROL

**Rotazione a Sinistra di un bit**

**Funzione:**

**Formato:**

| 011bbb10 | ADDR | ADDR |
|---|---|---|

**Descrizione:**

I contenuti dell'indirizzo specificato (accumulatore o memoria) sono ruotati a sinistra di una posizione. Il contenuto del carry va nel bit 0. Il bit 7 pone un nuovo valore nel carry. Questa è una rotazione a 9 bit.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

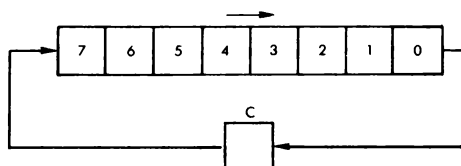
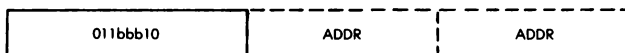
| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS, X | ABS, Y | (IND, X) | (IND), Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | 2A | 2E | 26 | | 3E | | | | 36 | | | |
| BYTES | | 1 | 3 | 2 | | 3 | | | | 2 | | | |
| CICLI | | 2 | 6 | 5 | | 7 | | | | 6 | | | |
| bbb | | 010 | 011 | 001 | | 111 | | | | 101 | | | |

**Flag:**

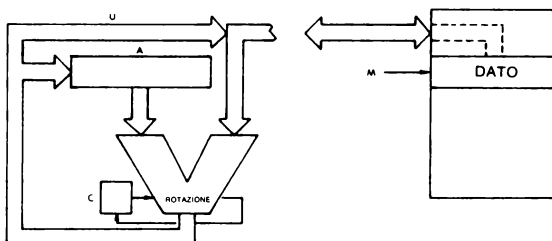
## Codici di Istruzione

# ROR Rotazione a Destra di un bit

**Attenzione:** Questa istruzione può non essere disponibile sui 6502 più vecchi. Inoltre essa può esistere ma non essere elencata.

**Funzione:**

**Formato:** 

| 011bbb10 | ADDR | ADDR |
|---|---|---|

**Descrizione:**

I contenuti dell'indirizzo specificato (accumulatore o memoria) sono ruotati a sinistra di una posizione di bit.. Il carry va nel bit 7. Il bit 0 pone un nuovo valore nel carry. Questa è una rotazione a 9 bit.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS. X | ABS. Y | (IND. X) | (IND), Y | PAGINA 0 X | PAGINA 0 Y | RELATIVE | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | 6A | 6E | 66 | | 7E | | | | 76 | | | |
| BYTES | | 1 | 3 | 2 | | 3 | | | | 2 | | | |
| CICLI | | 2 | 6 | 5 | | 7 | | | | 6 | | | |
| bbb | | 010 | 011 | 001 | | 111 | | | | 101 | | | |

**Flag:**

| N | V | B | D | I | Z | C |
|---|---|---|---|---|---|---|
| ● | | | | | ● | ● |

## Codici di Istruzione:

# RTI Ritorno da Interrupt

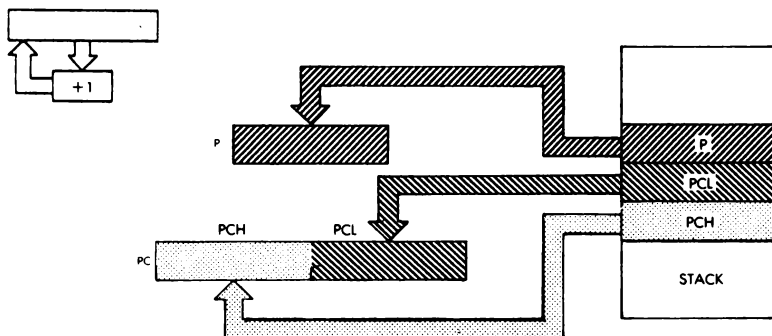
**Funzione:**

P ← (STACK)
S ← (S) + 1
PCL ← (STACK)
S ← (S) + 1
PCM ← (STACK)
S ← (S) + 1

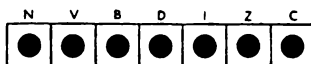
**Formato:**

**Descrizione:**

Ri-immagazzina il registro di stato ed il Contatore di Programma (PC) che erano conservati nello stack. Aggiusta il puntatore dello stack.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = 40, byte = 1, cicli = 6

**Flag:**

# RTS — Ritorno da Subroutine

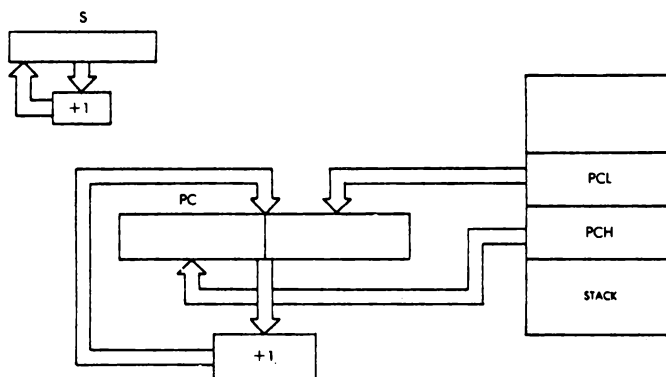
**Funzione:**

PCL ← (STACK)
S ← (S) + 1
PCH ← (STACK)
S ← (S) +
PC ← (PC + 1)

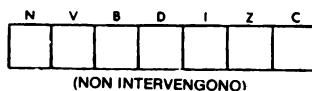
**Formato:**

**Descrizione:**

Ri-immagazzina il Contatore di Programma dello stack e lo incrementa di uno. Regola il puntatore dello stack.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = 60, byte = 1, cicli = 6

**Flag:**

# SBC

**Sottrae con Carry**

**Funzione:**

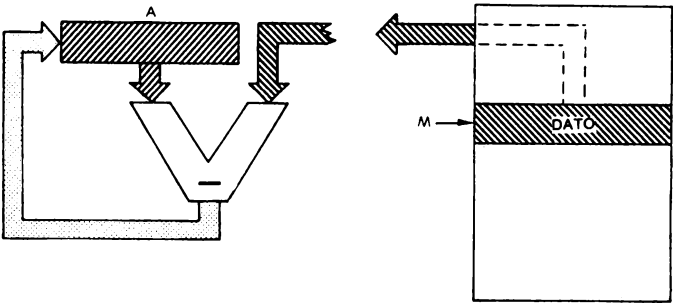
A ← (A) — DATO — C (C è il prestito)

**Formato:**

| 111bbb01 | ADDR/DATO | ADDR |
|---|---|---|

**Descrizione:**

Sottrae dall'accumulatore il dato all'indirizzo specificato, con prestito. Il risultato rimane in A. Nota: SEC è utilizzato per una sottrazione senza prestito.

SBC può essere utilizzato in modo decimale o binario, in funzione del bit D del registro di stato.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS. X | ABS. Y | (IND. X) | (IND), Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | | ED | E5 | E9 | FD | F9 | E1 | F1 | F5 | | | |
| BYTES | | | 3 | 2 | 2 | 3 | 3 | 2 | 2 | 2 | | | |
| CICLI | | | 4 | 3 | 2 | 4* | 4* | 6 | 5* | 4 | | | |
| bbb | | | 011 | 001 | 010 | 111 | 110 | 000 | 100 | 101 | | | |

*: PIU' 1 CICLO SE SI SUPERA LA PAGINA

**Flag:**

## Codici di Istruzione

*: PIU' 1 CICLO SE SI SUPERA LA PAGINA

# SEC　　　　　Set Carry

**Funzione:**

C ← 1

**Formato:** 00111000

**Descrizione:**

Il bit carry viene posto ad 1. Questa istruzione è utilizzata prima di SBC per eseguire una sottrazione senza carry.

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = 38, byte = 1, cicli = 2

**Flag:**

# SED Predisponi il modo decimale

**Funzione:**

D ← 1

**Formato:** 11111000

**Descrizione:**

Il bit decimale del registro di stato è posto ad 1. Quando esso è 0 il modo è binario. Quando esso è 1 il modo è decimale per ADC ed SBC.

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = F8, byte = 1, cicli = 2

**Flag:**

| N | V | B | D | I | Z | C |
|---|---|---|---|---|---|---|
| | | | 1 | | | |

# SEI    Set Disabilitazione Interrupt

**Funzione:**

I ← 1

**Formato:** 01111000

**Descrizione:**

La maschera di interrupt è posta ad 1. Viene utilizzata durante un interrupt oppure un ripristino del sistema.

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = 78, byte = 1, cicli = 2

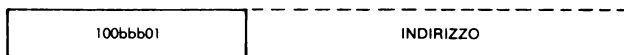
**Flag:**

# STA    Immagazzina l'accumulatore nella memoria

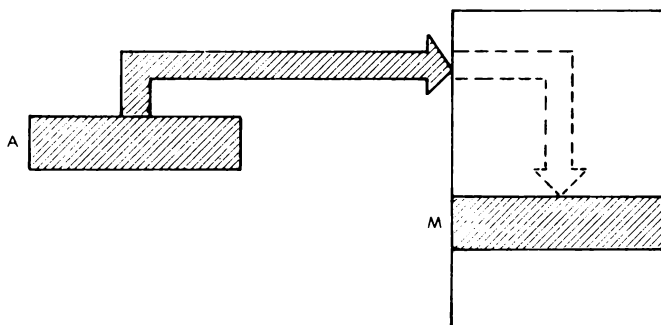
**Funzione:**

M ← (A)

**Formato:**

| 100bbb01 | INDIRIZZO |
|---|---|

**Descrizione:**

I contenuti di A vengono ricopiati alla locazione di memoria specificata. I contenuti di A non vengono cambiati.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULATORE | ASSOLUTO | PAGINA 0 | IMMEDIATO | ABS, X | ABS, Y | (IND, X) | (IND), Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | | 8D | 85 | | 9D | 99 | 81 | 91 | 95 | | | |
| BYTES | | | 3 | 2 | | 3 | 3 | 2 | 2 | 2 | | | |
| CICLI | | | 4 | 3 | | 5 | 5 | 6 | 6 | 4 | | | |
| bbb | | | 011 | 001 | | 111 | 110 | 000 | 100 | 101 | | | |

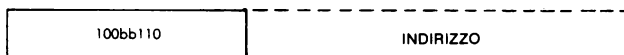
**Flag:**

## Codici di Istruzione:

# STX Immagazzina X nella memoria

**Funzione:**
M ← (X)

**Formato:**

| 100bb110 | INDIRIZZO |
|---|---|

**Descrizione:**

Copia i contenuti del registro indice X nella locazione di memoria specificata. I contenuti di X rimangono invariati.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULAT. | ASSOLUTO | PAGINA-0 | IMMEDIATO | ABS. X | ABS. Y | (IND. X) | (IND), Y | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | | 8E | 86 | | | | | | | 96 | | |
| BYTES | | | 3 | 2 | | | | | | | 2 | | |
| CICLI | | | 4 | 3 | | | | | | | 4 | | |
| bb | | | 01 | 00 | | | | | | | 10 | | |

**Flag:**

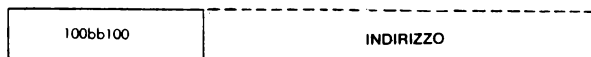
**Codici di Istruzione:**

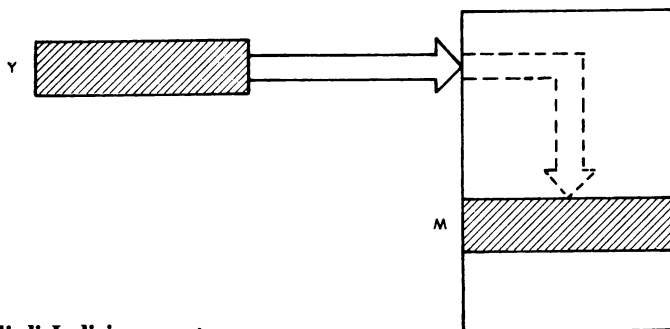
# STY　　Immagazzina Y nella memoria

**Funzione:**

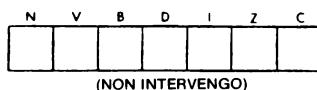
M ← (Y)

**Formato:**

| 100bb100 | INDIRIZZO |
|---|---|

**Descrizione:**

Copia i contenuti del registro indice Y nella locazione di memoria specificata. I contenuti di Y rimangono invariati.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

| | IMPLICATO | ACCUMULATORE | ASSOLUTO | PAGINA 0 | IMMEDIATO | ABS. X | ABS. Y | (IND. X) | (IND. Y) | PAGINA 0 X | PAGINA 0 Y | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | | | 8C | 84 | | | | | | 94 | | | |
| BYTES | | | 3 | 2 | | | | | | 4 | | | |
| CICLI | | | 4 | 3 | | | | | | 4 | | | |
| bb | | | 01 | 00 | | | | | | 10 | | | |

**Flag:**

**Codici di Istruzione:**

# TAX

**Trasferisce A in X**

**Funzione:**

X ← (A)

**Formato:** 10101010

**Descrizione:**

Copia i contenuti dell'accumulatore in X. I contenuti di A rimangono invariati.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = AA, byte = 1, cicli = 2

**Flag:**

# TAY

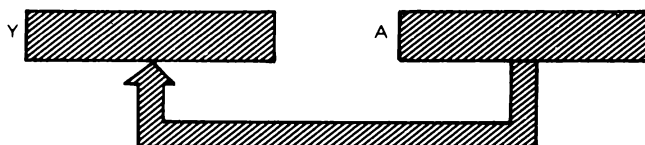
**Trasferisce l'accumulatore in Y**

**Funzione:**
Y ← (A)

**Formato:** 10101000

**Descrizione:**

Trasferisce i contenuti dell'accumulatore nel registro Y. I contenuti di A rimangono invariati.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = A8, byte = 1, cicli = 2

**Flag:**

## TSX **Trasferisce S in X**

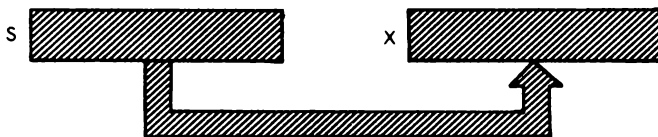
**Funzione:**
X ← (S)

**Formato:**

| 10111010 |
|---|

**Descrizione:**

I contenuti del puntatore dello stack S sono trasferiti nel registro indice X. I contenuti di S rimangono invariati.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = BA, byte = 1, cicli = 2

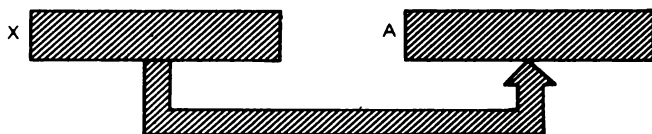
**Flag:**

# TXA Trasferisce X nell'Accumulatore

**Funzione:**
A ← (X)

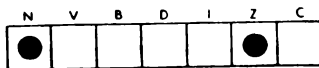
**Formato:** 10001010

**Descrizione:**
I contenuti del registro indice X sono trasferiti nell'accumulatore. I contenuti di X sono invariati.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = 8A, byte = 1, cicli = 2

**Flag:**

# TXS

**Trasferisce X in S**

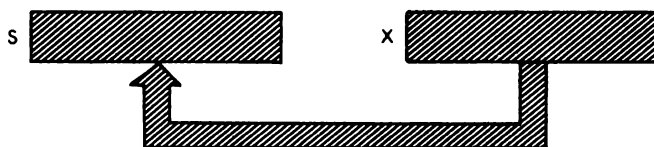
**Funzione:**

S ← (X)

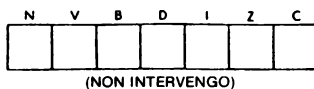
**Formato:** 10011010

**Descrizione:**

I contenuti del registro indice X sono trasferiti nel puntatore dello stack. I contenuti di X sono invariati.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = 9A, byte = 1, cicli = 2

**Flag:**

# TYA **Trasferisce Y in A**

**Funzione:**

A ← (Y)

**Formato:** 10011000

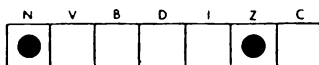
**Descrizione:**

I contenuti del registro indice Y sono trasferiti nell'accumulatore. I contenuti di Y sono invariati.

**Percorso dei Dati:**

**Modi di Indirizzamento:**

Soltanto implicato:
HEX = 98, byte = 1, cicli = 2

**Flag:**

CAPITOLO 5

# TECNICHE DI INDIRIZZAMENTO

## INTRODUZIONE

Questo capitolo presenterà la teoria generale dell'indirizzamento con le varie tecniche che sono state sviluppate per facilitare il recupero dei dati. Nel secondo paragrafo saranno analizzati in modi specifici di indirizzamento che sono disponibili nel 6502 con i loro vantaggi ed i loro limiti, dove esistono. Infine per familiarizzare il lettore con le varie possibilità di compromesso tra le diverse tecniche di indirizzamento si studieranno programmi specifici di applicazione.

Poichè il 6502 non ha registri a 16 bit, tranne il contatore di programma, che può essere impiegato per specificare un indirizzo, è necessario che l'utente del 6502 conosca i vari modi di indirizzamento ed, in particolare, l'impiego dei registri indice. I modi di recupero complessi, come una combinazione dell'indiretto ed indicizzato possono essere omessi a questo stadio iniziale. Comunque tutti i modi di indirizzamento sono utili per sviluppare programmi per questo microprocessore. Si studieranno ora le varie alternative disponibili.

## MODI DI INDIRIZZAMENTO

*L'indirizzamento* fa riferimento alle specifiche all'interno di una istruzione della locazione dell'operando su cui interviene l'istruzione stessa. Verranno ora esaminati i metodi principali.

### Indirizzamento Implicito

Le istruzioni che operano esclusivamente su registri normalmente utilizzano *l'indirizzamento implicito.* Questo è illustrato in Figura 5-1. Un'istruzione implicita deriva il suo nome dal fatto che essa non contiene specificamente l'indirizzo dell'operando su cui opera. Invece il suo codice operativo specifica uno o più registri (normalmente l'accumulatore od anche qualsiasi altro registro/i). Poichè i registri interni normalmente sono poco numerosi (diciamo un massimo di 8) questo richiederà un piccolo numero di bit. Per esempio tre bit dentro l'istruzione punte-

Figura 5.1: Indirizzamento

ranno da 1 ad 8 registri interni. Tali istruzioni possono perciò essere normalmente codificate all'interno di 8 bit. Questo è un vantaggio importante poichè un'istruzione ad 8 bit normalmente viene eseguita più velocemente di qualsiasi istruzione a due o tre byte.

Un esempio di istruzione implicita del 6502 è TXA che specifica "trasferisce i contenuti di A ad X".

### Indirizzamento Immediato

L'indirizzamento immediato è illustrato in Figura 5-1. Il codice operativo è seguito da un letterale ad 8 o 16 bit (una costante). Questo tipo di istruzione è necessario per esempio per caricare un valore ad 8 bit. Se il microprocessore è equipaggiato con registri a 16 bit può essere necessario caricare letterali a 16 bit. Questo dipende dall'architettura interna del processore. Un esempio di un'istruzione immediata è: ADC # 0.

La seconda parola di questa istruzione contiene il letterale "0" che è sommato all'accumulatore.

### Indirizzamento Assoluto

L'indirizzamento assoluto è il modo in cui i dati sono normalmente recuperati dalla memoria, dove un codice operativo è seguito da un indirizzo a 16 bit. L'indirizzamento assoluto perciò richiede istruzioni di 3 bit. Un esempio di indirizzamento assoluto è: STA $ 1234.

Questa istruzione specifica che i contenuti dell'accumulatore devono essere memorizzati alla locazione di memoria "1234" esadecimale.

Lo svantaggio dell'indirizzamento assoluto è di richiedere un'istruzione di 3 byte. Per migliorare l'efficienza del microprocessore può essere reso disponibile un altro modo di indirizzamento nel quale per l'indirizzo viene utilizzata una sola parola: indirizzamento diretto.

### Indirizzamento Diretto

In questo modo di indirizzamento il codice operativo è seguito da un indirizzo ad 8 bit. Questo è illustrato in Figura 5-1. Il vantaggio di questo approccio è di richiedere solo 2 byte invece dei 3 dell'indirizzamento assoluto. Lo svantaggio è la limitazione di tutti gli indirizzamenti all'interno di questo modo per indirizzare da 0 a 255. Questa è la Pagina 0. Questo è anche chiamato l'indirizzamento breve od indirizzamento in Pagina 0. Ogni volta che è disponibile l'indirizzamento breve, l'indirizzamento assoluto è spesso chiamato *indirizzamento esteso* per contrasto.

### Indirizzamento Relativo

Le normali istruzioni di salto o diramazione richiedono 8 bit per il

codice operativo più l'indirizzo a 16 bit al quale deve passare l'esecuzione del programma. Come nell'esempio precedente questo ha l'inconveniente di richiedere tre parole cioè 3 cicli di memoria. Per fornire una diramazione più efficiente l'indirizzamento relativo utilizza un formato di sole due parole. La prima parola è la specifica della diramazione, normalmente assieme al test che si sta realizzando. La seconda parola è uno spostamento. Poichè lo spostamento può essere positivo o negativo un'istruzione di diramazione relativa consente una diramazione diretta fino a 128 locazioni (7 bit) oppure una diramazione inversa fino a 128 locazioni (più o meno 1 in dipendenza delle convenzioni). Poichè la maggior parte dei cicli tendono ad essere brevi la diramazione relativa può essere utilizzata quasi sempre e si risolve in un significativo miglioramento di esecuzione di tali routine brevi. Come esempio è già stata utilizzata l'istruzione BCC che specifica "operazione diramazione se carry è zero" alla locazione all'interno di 127 parole dall'istruzione di diramazione stessa.

### Indirizzamento Indicizzato

L'indirizzamento indicizzato è una tecnica specificamente pratica per accedere successivamente agli elementi di un blocco o di una tabella. Questo sarà illustrato mediante esempi nel corso di questo capitolo. Il principio dell'indirizzamento indicizzato è che l'istruzione specifica sia un registro indice che un indirizzo. Nello schema più generale i contenuti del registro sono sommati all'indirizzo per fornire l'indirizzo finale. In questo modo l'indirizzo potrebbe essere poi utilizzato per accedere successivamente a tutti gli elementi di una tabella in modo efficiente. In pratica esistono spesso restrizioni e si può limitare la dimensione del registro indice o la dimensione dell'indirizzo o campo di spostamento.

### Pre-indicizzazione e Post-indicizzazione

Si possono distinguere due modi di indicizzazione. La pre-indicizzazione è il modo di indicizzazione usuale dove l'indirizzo finale è la somma di uno spostamento od indirizzo o dei contenuti del registro indice.

La post-indicizzazione tratta i contenuti del campo di spostamento come *l'indirizzo* dello spostamento effettivo, piuttosto che lo spostamento stesso. Questo è illustrato in Figura 5-2. Nella post-indicizzazione l'indirizzo finale è la somma dei contenuti del registro indice più i contenuti della parola di memoria *designata dal campo di spostamento.* Questa caratteristica utilizza infatti una combinazione dell'indirizzamento indiretto e della pre-indicizzazione. Si noti che non è stato ancora

definito l'indirizzamento indiretto. È quello che si farà immediatamente.

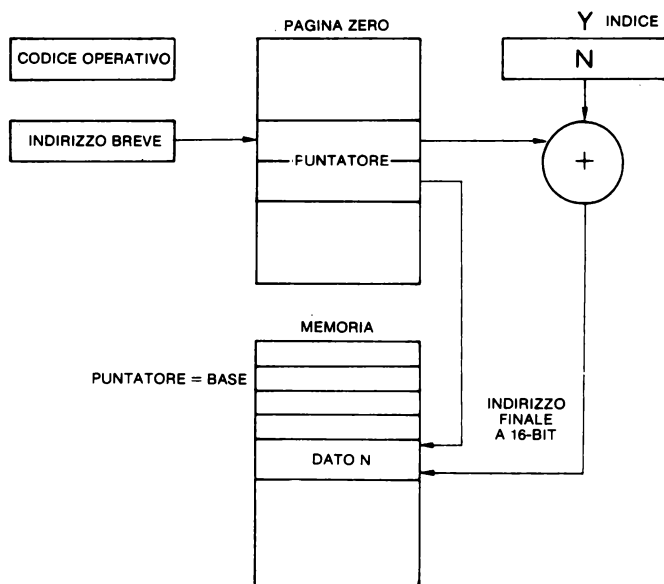



Figura 5.2: Indirizzamento indicizzato indiretto

## Indirizzamento Indiretto

È già stato visto il caso in cui due subroutine devono scambiarsi una grande quantità di dati immagazzinati nella memoria. Più in generale diversi programmi o diverse subroutine, possono richiedere di accedere a blocchi comuni di informazioni. Per preservare la generalità del programma è desiderabile non mantenere tale blocco ad una fissata locazione di memoria. In particolare la dimensione di questo blocco può crescere o diminuire dinamicamente e può risiedere in varie aree di memoria, in funzione della sua dimensione. Sarebbe perciò impraticabile in generale cercare di avere accesso a questo blocco impiegando l'indirizzamento assoluto.

La soluzione a questo problema sta nel depositare l'indirizzo di partenza del blocco ad una fissata locazione di memoria. Questo è analogo alla situazione in cui diverse persone devono entrare in una casa ed esiste solo una chiave. Per convenzione la chiave della casa sarà nascosta sotto il vaso. Ogni utilizzatore conoscerà dove guardare (sotto il vaso) per trovare la chiave della casa (ovvero per trovare l'indirizzo della lista

richiesta, per analogia). L'indirizzamento indiretto perciò utilizza un codice operativo di 8 bit seguito da un indirizzo a 16 bit. Questo indirizzo è utilizzato semplicemente per recuperare una parola dalla memoria. Normalmente sarà una parola a 16 bit (nel nostro caso due byte) all'interno della memoria. Questo è illustrato dalla figura 5-3). I due byte all'indirizzo specificato $A_1$. $A_2$ sono quindi interpretati come indirizzo effettivo dei dati ai quali si desidera accedere.

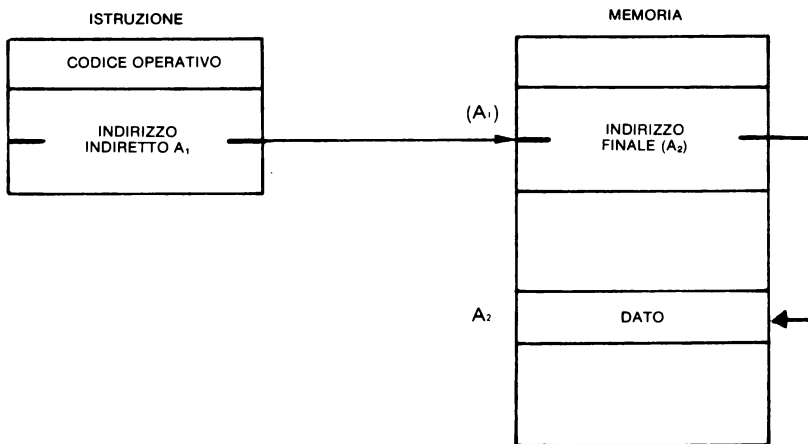



Figura 5.3: Indirizzamento indiretto

L'indirizzamento indiretto è particolarmente conveniente tutte le volte che sono utilizzati i puntatori. Varie aree del programma possono perciò fare riferimento a questi puntatori per accedere convenientemente ed elegantemente ad una parola o ad un blocco di dati.

### Combinazione dei Modi

I precedenti modi di indirizzamento possono essere combinati. In particolare sarebbe possibile, in uno schema di indirizzamento completamente generale utilizzare molti livelli di indirizzamento indiretto. L'indirizzo $A_2$ potrebbe essere interpretato come un ulteriore indirizzo indiretto e così via.

L'indirizzamento indicizzato può essere anche combinato con l'accesso indiretto. Questo consente l'accesso efficiente alla parola n di un blocco di dati forniti una volta che si conosce dove è indirizzato il puntatore all'indirizzo di partenza.

Si è così divenuti familiari con tutti i modi di indirizzamento usuali che possono essere disponibili in un sistema. La maggior parte dei sistemi a microprocessore, a causa della limitazione sulla complessità della MPU, che deve essere realizzata all'interno di un singolo chip, non forniscono tutti i modi possibili ma soltanto un piccolo sottinsieme di questi. Il 6502 fornisce un sottinsieme non comunemente largo di possibilità. Si esamineranno ora queste possibilità.

## MODI DI INDIRIZZAMENTO DEL 6502

### Indirizzamento Implicato (6502)

L'indirizzamento implicato è utilizzato da un'istruzione a singolo byte che opera sui registri interni. Ogni volta che le istruzioni implicite operano esclusivamente sui registri interni, queste richiedono soltano due cicli di clock per essere eseguite. Ogni volta che esse accedono alla memoria richiedono tre cicli.

Le istruzioni che operano esclusivamente sui registri interni sono CLC, CLD, CLI, CLV, DEX, DEY, INX, INY, NOP, SEC, SED, SEI, TAX, TAY, TSX, TXA, TXS, TYA.

Le istruzioni che richiedono l'accesso alla memoria sono: BRK, PHA, PHP, PLA, PLP, RTI, RTS.

Queste istruzioni sono state descritte al capitolo precedente ed il loro modo di operare dovrebbe essere chiaro.

### Indirizzamento Immediato (6502)

Poichè il 6502 ha soltanto registri di lavoro ad 8 bit (il PC non è un registro di lavoro) l'indirizzamento immediato nel caso del 6502 è limitato alle costanti ad 8 bit. Tutte le istruzioni nel modo ad indirizzamento immediato sono perciò lunghe due byte. Il primo byte contiene il codice operativo ed il secondo byte contiene la costante od il letterale che deve essere caricato nel registro od utilizzato in congiunzione con uno dei registri per un'operazione aritmetica o logica.

Le istruzioni che utilizzano questo modo di indirizzamento sono: ADC, AND, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA, SBC.

### Indirizzamento Assoluto (6502)

Per definizione l'indirizzamento assoluto richiede 3 byte. Il primo byte è il codice operativo ed i due byte successivi sono l'indirizzo a 16 bit specificante la locazione dell'operando. Eccetto il caso di un salto assoluto, questo modo di indirizzo richiede quattro cicli.

Le istruzioni che possono utilizzare l'indirizzamento assoluto sono: ADC, AND, ASL, BIT, CMP, CPX, CPY, DEC, EOR, INC, JMP, JSR, LDA, LDX, LDY, LSR, ORA, ROL, ROR, SBC, STA, STX, STY.

### Indirizzamento in Pagina Zero (6502)

Per definizione l'indirizzamento in pagina zero richiede due byte: il primo è per il codice operativo: il secondo è per l'indirizzo breve ad 8 bit.

L'indirizzamento in pagina zero richiede tre cicli. Poichè l'indirizzamento in pagina zero offre significativi vantaggi in velocità in virtù del codice più breve, esso dovrebbe essere utilizzato dovunque possibile. Questo richiede un'attenta gestione della memoria da parte del programmatore. Parlando in generale le prime 256 locazioni di memoria possono essere viste come un set di registri di lavoro per il 6502. Qualsiasi istruzione sarà essenzialmente eseguita su questi 256 "registri" in appena tre cicli. Questo spazio dovrebbe perciò essere attentamente riservato per i dati essenziali che necessitano di essere recuperati ad alta velocità.

Le istruzioni che possono utilizzare l'indirizzamento in pagina zero sono quelle che possono utilizzare l'indirizzamento assoluto eccetto JMP e JSR (che richiedono un indirizzo a 16 bit).

La lista delle istruzioni consentite è quindi: ADC, AND, ASL, BIT, CMP, CPX, CPY, DEC, EOR, INC, LDA, LDX, LDY, LSR, ORA, ROL, ROR, SBC, STA, STX, STY.

### Indirizzamento Relativo (6502)

Per definizione l'indirizzamento relativo utilizza due byte. Il primo è un'istruzione di salto mentre il secondo specifica lo spostamento ed il suo segno. Per differenziare questo modo dall'istruzione di salto esse sono indicate qui come *diramazione.* Le diramazioni, nel caso del 6502, utilizzano sempre il modo relativo. I salti utilizzano sempre il modo assoluto (più naturalmente gli altri sotto-modi che possono essere combinati con queste come Indicizzato ed Indiretto). Da un punto di vista del timing questa istruzione dovrebbe essere esaminata con cautela. Ogni volta che un test è soddisfatto, cioè ogni volta che non c'è diramazione, questa istruzione richiede solo due cicli. Questo perchè la successiva istruzione da eseguire è puntata dal contatore di programma. Invece ogni volta che il test è soddisfatto, questa istruzione richiede tre cicli: deve essere calcolato un nuovo effettivo indirizzo. L'aggiornamento del contatore di programma richiede un ulteriore ciclo. Comunque se si verifica una diramazione oltre ai confini di una pagina, un ulteriore

aggiornamento è necessario per il contatore di programma e la lunghezza effettiva dell'istruzione diviene di quattro cicli.

L'utente non deve preoccuparsi da un punto di vista logico dell'attraversamento della frontiera di una pagina. Si prende cura di questo l'hardware. Comunque, poichè un ulteriore riporto o prestito è generato ogni volta che si attraversa la frontiera di una pagina, il tempo di esecuzione della diramazione cambierà. Se questa diramazione fa parte di un esatto ciclo di timing occorre fare attenzione.

Un buon assemblatore dirà normalmente al programmatore, all'istante in cui il programma è assemblato, se una diramazione provoca l'attraversamento della frontiera di una pagina nel cui caso il timing può essere critico.

Ogni volta che non si è sicuri se si verificherà una diramazione si deve tener conto che alcune volte la diramazione richiederà due cicli ed altre volte tre. Spesso viene calcolato un tempo medio.

Le sole istruzioni che realizzano un indirizzamento relativo sono le situazioni di diramazione. Ci sono 8 istruzioni di diramazione che operano il test di ciascun flag all'interno del registro di stato per i valori "0" ed "1", più l'istruzione BIT. La lista è: BCC, BEQ, BMI, BNE, BPL, BVC, BVS.

### Indirizzamento Indicizzato (6502)

Il 6502 non fornisce una capacità completamente generale ma soltanto una limitata. Esso è equipaggiato con due registri indice. Comunque questi registri sono limitati ad 8 bit. I contenuti di un registro indice sono sommati al campo indirizzo dell'istruzione. Normalmente il registro indice è utilizzato come contatore per accedere agli elementi successivi di un blocco o di una tabella. Questo perchè sono disponibili istruzioni specializzate per incrementare o decrementare ciascuno dei registri indice separatamente. Inoltre esistono due istruzioni specializzate per confrontare i contenuti dei registri indice con una locazione di memoria, un'importante possibilità per l'effettivo impiego dei registri indice per operare il test rispetto ai limiti consentiti.

In pratica, poichè la maggior parte delle tabelle dell'utente sono normalmente più corte di 256 parole la limitazione dei registri indice ad 8 bit è normalmente una limitazione non significativa.

Il modo di indirizzamento indicizzato può essere utilizzato non solo con l'indirizzamento assoluto regolare, cioè con un campo di indirizzo a 16 bit, ma anche con il modo di indirizzamento in pagina zero, cioè con i campi indirizzo ad 8 bit.

C'è soltanto una restrizione. Il registro X può essere utilizzato da

entrambi i tipi di indirizzamento. Invece il registro Y consente solo l'indirizzamento *assoluto* indicizzato e non quello indicizzato in *pagina zero*. (Eccetto per le istruzioni LDX ed STX che possono essere modificate dal registro Y).

L'indirizzamento indicizzato assoluto richiederà quattro cicli, se non si attraversa la frontiera di una pagina, nel cui caso saranno richiesti cinque cicli.

Le istruzioni indicizzate assolute possono utilizzare sia il registro X che Y per fornire il campo di spostamento. La lista delle istruzioni che possono utilizzare questo modo sono:

—con X: ADC, AND, ASL, CMP, DEC, EOR, INC, LDA, LDY, LSR, ORA, ROL, ROR, SBC, STA (non STY).

—con Y: ADC AND, CMP, EOR, LDA, LDX, ORA, SBC, STA (non ASL, DEC, LSR, ROL, ROR).

Nel caso di indirizzamento indicizzato in pagina zero il registro X è il solo registro di spostamento consentito. Le istruzioni consentite sono: ADC, AND, ASL, CMP,DEC, EOR, INC, LDA, LDY, LSR, ORA, ROL, ROR, SBC, STA, STY.

### Indirizzamento Indiretto (6502)

Il 6502 non ha la capacità di indirizzamento indiretto completamente generale. Esso limita il campo indirizzo ad 8 bit. In altre parole tutti gli indirizzamenti indiretti utilizzano il sotto-modo di indirizzamento indiretto in pagina zero. L'indirizzo effettivo su cui opera il codice operativo sono quindi i 16 bit specificati dall'indirizzo in pagina zero dell'istruzione. Inoltre non si può utilizzare un indiretto di ordine superiore. Questo significa che un indirizzo recuperato dalla pagina zero deve essere usato come tale e non può essere utilizzato come ulteriore indirezione.

Infine tutti gli accessi indiretti devono essere indicizzati, eccetto JMP. Per imparzialità si potrebbe notare che pochissimi microprocessori forniscono qualsiasi indirizzamento completamente indiretto. Inoltre è possibile realizzare un indirizzamento indiretto più generale utilizzando una definizione macro.

Sono possibili due modi di indirizzamento indiretto: indirizzamento indiretto indicizzato ed indirizzamento indicizzato indiretto (eccetto JMP che utilizza l'indiretto puro).

*Indirizzamento Indiretto Indicizzato*

Questo modo somma i contenuti del registro indice X all'indirizzo in pagina zero per calcolare l'indirizzo finale a 16 bit. Questo è un modo efficiente per recuperare uno dei diversi dati possibili per dati puntati mediante puntatori il cui numero è contenuto nel registro indice X. Questo è illustrato in Figura 5-4.

In questa illustrazione la pagina zero contiene una tabella di puntatori. Il primo puntatore è all'indirizzo A che fa parte dell'istruzione. Se i contenuti di X sono 2N allora questa istruzione accederà al numero N di puntatori di questa tabella e recupererà i dati puntati.

L'indirizzamento indiretto indicizzato richiede 6 cicli. Esso è naturalmente meno efficiente come impiego di tempo di qualsiasi modo di indirizzamento diretto. Il suo vantaggio è la flessibilità che può risultare nella codifica ovvero il miglioramento globale di velocità.

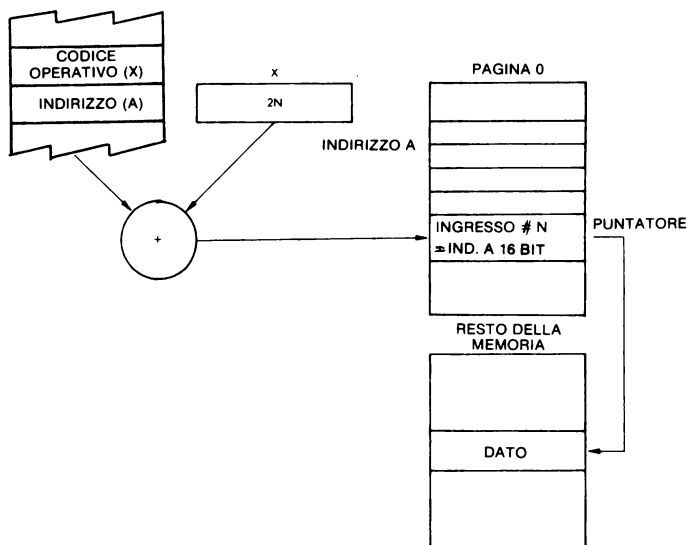



Figura 5.4: Indirizzamento indicizzato

Le istruzioni consentite sono: ADC, AND, CMP, EOR, LDA, ORA, SBC, STA.

*Indirizzamento Indicizzato Indiretto*

Questo corrisponde al meccanismo della post-indicizzazione che è stato descritto al paragrafo precedente. In quella sede l'indicizzazione

era eseguita dopo l'indirezione, piuttosto che prima. In alte parole l'indirizzo corto che è parte delle istruzioni è utilizzato per accedere ad un puntatore a 16 bit in pagina zero. I contenuti del registro indice Y sono quindi sommati come uno spostamento a questo puntatore. Il dato finale è quindi recuperato.

In questo caso il puntatore contenuto in pagina zero indica la base di una tabella nella memoria. Il registro Y fornisce uno spostamento. Questo è un vero indice all'interno di una tabella. Questa istruzione è particolarmente potente per far riferimento all'ennesimo elemento di una tabella, premesso che l'indirizzo di partenza della tabella è conservato in pagina zero. Si può fare questo in soli due byte.

Le istruzioni consentite sono: ADC, CMP, EOR, LDA, ORA, SBC, STA.

*Eccezione: Istruzione di Salto*

L'istruzione salto può usare l'assoluto indiretto. È la sola istruzione che può usare questo modo.

## UTILIZZAZIONE DEI MODI DI INDIRIZZAMENTO DEL 6502

### Indirizzamento Lungo e Breve

Sono già state utilizzate le istruzioni di diramazione in vari programmi tra quelli sviluppati. Essi sono auto esplicativi. Una domanda interessante è la seguente: cosa si può fare se il range consentito per la diramazione non è sufficiente per richieste particolari? Una semplice soluzione è di utilizzare la cosiddetta *diramazione lunga.* Questa è semplicemente una diramazione alla locazione che contiene una specifica di salto:

```
BCC + 3                      OPERA LA DIRAMAZIONE ALL'INDIRIZZO
                             EFFETTIVO
                             + 3 SE C'È ZERO
JMP FAR                      ALTRIMENTI SALTA A FAR
(ISTRUZIONE SUCCESSIVA)
```

Il precedente programma di due istruzioni si risolverà nella diramazione alla locazione FAR ogni volta che il carry è zero. Questo risolve il problema della diramazione lunga. Si considerino perciò ora i modi di indirizzamento più complessi cioè l'indicizzazione e l'indirezione.

### Utilizzazione dell'indicizzazione per l'accesso di blocchi sequenziali

L'indicizzazione è innanzitutto utilizzata per indirizzare locazioni

successive all'interno di una tabella. La restrizione consiste nel fatto che il massimo spostamento deve essere minore di 256 cosicchè esso possa risiedere in un registro indice ad 8 bit.

Si è imparato a controllare il carattere "*". Ora si cercherà in una tabella di 100 elementi il carattere "*". L'indirizzo di partenza di questa tabella è chiamata BASE. La tabella ha soltanto 100 elementi. Questi sono minori di 256 e si può quindi utilizzare un registro indice. Il programma appare come segue:

```
SEARCH          LDX # 0
NEXT            LDA BASE, X
                CMP "*"
                BEQ STARFOUND
                INX
                CPX # 100
                BNE NEXT
NOTFOUND        ...
STARFOUND       ...
```

Il diagramma di flusso di questo programma appare in Figura 5-5. Si potrebbe facilmente verificare l'equivalenza tra il diagramma di flusso ed il programma. La logica del programma è abbastanza semplice. Il registro X è utilizzato per puntare all'elemento all'interno della tabella. La seconda istruzione del programma:

NEXT LDA BASE, X

utilizza l'indirizzamento indicizzato assoluto. Esso specifica che l'accumulatore deve essere caricato all'indirizzo BASE (indirizzo assoluto a 16 bit) più i contenuti di X. All'inizio i contenuti di X sono "0". Il primo elemento da accedere sarà quello all'indirizzo BASE. Si può vedere che dopo l'interazione successiva, X avrà il valore "1" e si accederà all'elemento sequenzialmente successivo della tabella, all'indirizzo BASE + 1.

La terza istruzione del programma CMP "*" confronta il valore del carattere che è stato letto nell'accumulatore con il codice di "*". La successiva istruzione opera il test dei risultati del confronto. Se è stato trovato un accordo si verifica una diramazione alla label STARFOUND:

BEQ STARFOUND

Altrimenti viene eseguita l'istruzione sequenzialmente successiva:

INX

Il contatore indice è incrementato di 1. Con riferimento al diagramma di flusso della Fig. 5-5, si trova esaminando la parte bassa di quest'ultimo che il valore del registro indice a questo punto deve essere controllato per assicurarsi di non oltrepassare i confini della tabella (in questo caso 100 elementi). Questo è realizzato dall'istruzione seguente:

CPX # 100

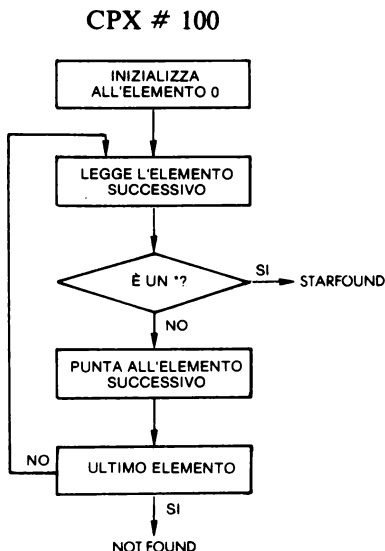



Figura 5.5: Ricerca di un carattere in una tabella

Questa istruzione confronta il valore del registro X col valore $ 100. Se il test non è soddisfatto si deve prelevare ancora il carattere successivo. Questo è quanto succede con:

BNE NEXT

Questa istruzione specifica una diramazione alla label NEXT se il test non è stato soddisfatto (la seconda istruzione del programma). Questo ciclo sarà eseguito finchè non è stato trovato un "*" oppure finchè non è stato raggiunto il valore dell'indice "100". Quindi sarà eseguita l'istruzione sequenzialmente successiva "NOT FOUND". Questo corrisponde al caso in cui non è stato trovato un "*".

Le azioni intraprese nei casi di "*" trovato o non trovato qui sono irrilevanti e dovrebbero essere specificate dal programmatore.

Si è così imparato ad utilizzare il modo di indirizzamento indicizzato

per accedere agli elementi successivi di una tabella. Si utilizzerà ora questa nuova abilità e si assumerà leggermente la difficoltà. Si svilupperà un programma di utilità notevole capace di copiare un blocco da un'area della memoria ad un'altra. Si assumerà inizialmente che il numero di elementi all'interno del blocco sia minore di 256 cosicchè sia possibile utilizzare il registro indice X. Quindi si considererà il caso generale in cui il numero di elementi del blocco sia maggiore di 256.

### Una Routine di Trasferimenti di Blocco per meno di 256 elementi

Si chiamerà "NUMBER" il numero di elementi del blocco da trasferire. Il numero è assunto essere minore di 256. BASE è l'indirizzo base del blocco. DESTINATION è la base dell'area di memoria dove si muoverà il blocco. L'algoritmo è abbastanza semplice: si muoverà una parola alla volta, mantenendo la traccia della parola che si sta muovendo, immagazzinando la sua posizione nel registro indice X. Il programma è il seguente:

```
            LDX   # NUMBER
NEXT        LDA   BASE, X
            STA   DEST, X
            DEX
            BNE   NEXT
```

Si esamini questo programma:

LDX # NUMBER

Questa riga del programma carica il numero N di parole da trasferire nel registro indice. L'istruzione successiva carica la parola # N del blocco all'interno dell'accumulatore e la terza istruzione la deposita nell'area di destinazione. Si veda la figura 5-6.

*ATTENZIONE:* questo programma lavorerà correttamente solo se il registro di base è assunto puntare proprio *sotto* il blocco come il registro di destinazione. Diversamente è richiesto un piccolo aggiustamento a questo programma.

Dopo che una parola è stata trasferita dall'origine all'area di destinazione il registro indice deve essere aggiornato. Questo è eseguito dall'istruzione DEX che lo decrementa. Quindi il programma opera semplicemente il test se X è stato decrementato a 0. Se si il programma termina. Diversamente esso cicla ancora ritornando alla locazione NEXT.

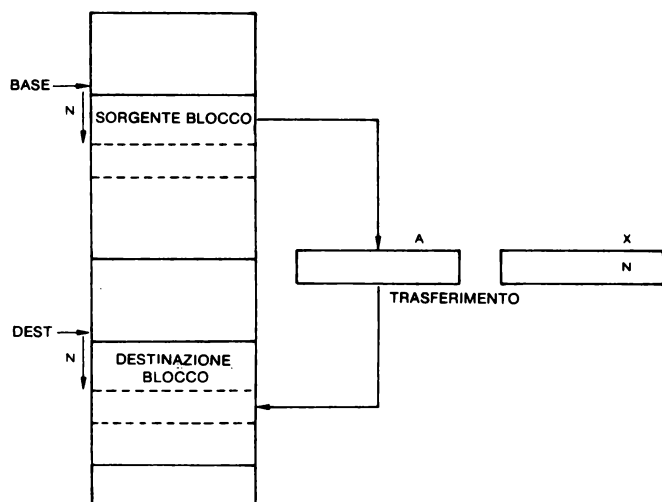



Figura 5.6: Organizzazione di memoria per il trasferimento di blocco generale

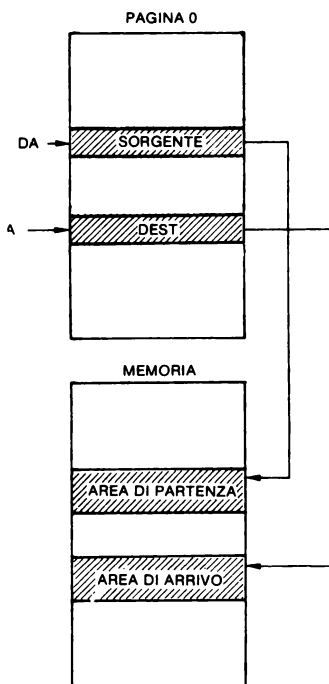



Figura 5.7: Mappa di memoria per un trasferimento di blocco generale

Si noterà che quando X = 0 il programma *non cicla.* Perciò esso non trasferirà la parola alla locazione BASE. L'ultima parola trasferita sarà quella a BASE + 1. Questo perchè è stato assunto che la base puntasse proprio *sotto* il blocco.

**Esercizio 5.1:** *Si modifichi il programma precedente assumendo che BASE e DEST puntino proprio al primo ingresso del blocco.*

Questo programma illustra anche l'uso dei contatori del ciclo. Si noterà che X è stato caricato con il valore *finale* quindi *decrementato* e verificato. A prima vista potrebbe sembrare più semplice iniziare col valore "0" in X e quindi incrementarlo fino a che esso raggiunge il massimo valore. Comunque per operare il test se X ha raggiunto il suo massimo valore sarebbe necessaria un'ulteriore istruzione (l'istruzione di confronto). Questo ciclo richiederebbe quindi 5 istruzioni invece di 4. Poichè questo programma di trasferimento sarà utilizzato normalmente per numeri elevati di parole, è significativamente importante ridurre il numero di istruzioni del ciclo. Questa è la ragione per cui, almeno per cicli brevi, il registro indice è normalmente *decrementato piuttosto che incrmentato.*

## Una Routine di Trasferimento di Blocco (più di 256 elementi)

Si consideri ora il caso generale di movimento di un blocco che può contenere più di 256 elementi. Non è possibile utilizzare un singolo registro indice ad 8 bit perchè insufficiente per immagazzinare un numero maggiore di 256. L'organizzazione della memoria per questo programma è illustrata in Figura 5-7. La lunghezza del blocco di memoria da trasferire richiede 16 bit e perciò è immagazzinato in memoria. La parte di ordine elevato rappresenta il numero di blocchi di 256 parole: "BLOCKS". Il resto è chiamato "REMAIN" ed è in numero di parole da trasferire dopo che tutti i blocchi sono stati trasferiti. L'indirizzo della sorgente a destinazione sarà alle locazioni di memoria FROM e TO rispettivamente. Si assumerà innanzi tutto che REMAIN sia zero cioè che si stiano trasferendo blocchi di 256 parole. Il programma è il seguente:

```
LDA  # SOURCELO
STA  FROM
LDA  # SOURCEHI
STA  FROM + 1       IMMAGAZZINA L'INDIRIZZO SORGENTE
```

```
        LDA  # DESTLO
        STA  TO
        LDA  # DESTHI
        STA  TO + 1        IMMAGAZZINA L'INDIRIZZO DEST
        LDX  # BLOCKS      QUANTI BLOCCHI
        LDY  # 0           DIMENSIONE BLOCCO
NEXT    LDA  (FROM), Y     LEGGE ELEMENTO
        STA  (TO), Y       LO TRASFERISCE
        DEY                AGGIORNA IL PUNTATORE DELLA PAROLA
        BNE  NEXT          FINITO?
NEXBLK  INC  FROM + 1      INCREMENTA IL PUNTATORE DEL BLOCO
        INC  TO + 1        LO STESSO
        DEX                CONTATORE BLOCCO
        BMI  DONE
        BNE  NEXT
        LDY  # REMAIN
        BNE  NEXT
```

L'indirizzo sorgente a 16 bit è immagazzinato dalle prime quattro istruzioni all'indirizzo di memoria "FROM". Le successive quattro istruzioni fanno la stessa cosa per la destinazione che è immagazzinata all'indirizzo "TO". Poichè si deve trasferire un numero di parole maggiore di 256 si utilizzeranno semplicemente due registri indice ad 8 bit.

L'istruzione successiva carica il registro X con il numero di blocchi che devono essere trasferiti. Questa è l'istruzione 9 del programma. L'istruzione successiva carica il valore 0 nel registro indice Y, per inizializzarlo al trasferimeno di 256 parole.

Si utilizzerà ora l'indirizzamento indiretto indicizzato. Si dovrebbe ricordare che l'indiretto indicizzato si risolverà prima in una indirezione all'interno della pagina zero quindi in un accesso indicizzato all'indirizzo a 16 bit specificato dal registro indice. Si osservi il programma:

NEXT LDA (FROM), Y

Questa istruzione carica l'accumulatore con i contenuti della locazione di memoria il cui indirizzo è la sorgente più i contenuti del registro indice Y.

Si osservi la Figura 5-7 per la mappa di memoria. Qui il contenuto del registro Y è inizialmente 0. "A" sarà perciò caricato dall'indirizzo di memoria "SOURCE". Si noti che qui, diversamente dall'esempio precedente, si assume che "SOURCE" è l'indirizzo della prima parola all'interno del blocco.

Utilizzando la stessa tecnica l'istruzione successiva depositerà i contenuti dell'accumulatore (la prima parola del blocco che si vuole trasferire) all'appropriata locazione di destinazione:

STA (TO), Y

Proprio come nel caso precedente si decrementa semplicemente il registro indice quindi si cicla 256 volte. Questo è realizzato dalle due istruzioni successive:

DEY

BNE NEXT

*Attenzione:* un artificio di programmazione viene qui utilizzato per una programmazione compatta. Il lettore attento noterà che il registro indice Y è *decrementato.* La prima parola ad essere trasferita sarà perciò la parola in posizione 0. Quella successiva sarà la parola 255. Questo perchè decrementando 0 si ottengono tutti uni nel registro (oppure 255). Il lettore dovrebbe anche accertare che qui non ci sono errori. Ogni volta che il registro Y decrementa a 0 *non si verificherà* un trasferimento. L'istruzione successiva da eseguire sarà: NEX BLK. Perciò saranno state trasferite esattamente 256 parole. Chiaramente questo stesso artificio potrebbe essere utilizzato nei vari programmi precedenti per scrivere un programma più breve.

Una volta trasferito un blocco completo si tratta semplicemente di puntare la pagina successiva all'interno del blocco originale e del blocco di destinazione. Questo si ottiene aggiungendo "1" alla parte di ordine più elevato dell'indirizzo della sorgente e destinazione. Questo è eseguito da due istruzioni successive del programma:

```
NEXBLK  INC    FROM + 1
        INC    TO + 1
```

Dopo avere incrementato il puntatore della pagina si controlla semplicemente se è stato trasferito il numero sufficiente di blocchi decrementando il blocco contatore contenuto in X. Questo è eseguito da:

DEX

Se tutti i blocchi sono stati trasferiti si esce dal programma mediante la diramazione alla locazione DONE:

BMI DONE

A questo punto si hanno due possibilità: X non decrementato a 0 oppure esattamente decrementato a zero. Se non è stato decrementato a 0 si ha la diramazione alla locazione NEXT:

BNE NEXT

Se è stato decrementato esattamente a 0 si ha il trasferimento delle parole specificate da REMAIN. Questa è l'ultima parte del trasferimento. Questo è seguito da:

LDY # REMAIN

che carica l'indice Y con il conteggio del trasferimento.
Quindi si ha la diramazione alla locazione NEXT:

BNE NEXT

Il lettore dovrebbe accertare che durante quest'ultimo ciclo dove è eseguita l'istruzione di diramazione a NEXT, la volta successiva si rientra a NEXBLK e quindi si uscirà da questo programma. Questo perchè l'indice X ha il valore 0 prima di entrare in NEXBLK. La terza istruzione di NEXBLK lo cambierà a — 1 e si uscirà a DONE.

**Somma di due Blocchi**

Questo esempio fornirà una semplice illustrazione dell'utilizzazione di un registro indice per l'addizione di due blocchi di meno di 256 elementi. Successivamente il programma che seguirà farà uso della caratteristica di indicizzazione indiretta per indirizzare i blocchi i cui indirizzi sono noti risiedere ad una data locazione, ma i cui indirizzi effettivi assoluti non sono noti. Il programma è il seguente:

```
BLKADD  LDY  # NBR — 1  ———  CARICA IL CONTATORE
NEXT    CLC
        LDA  PTR1, Y    ———  LEGGE L'ELEMENTO SUCCESSIVO
        ADC  PTR2, Y         LI SOMMA
        STA  PTR3, Y         IMMAGAZZINA IL RISULTATO
        DEY                  DECREMENTA IL CONTATORE
        BPL  NEXT            FINITO?
```

L'indice Y è utilizzato come contatore indice ed è caricato col numero di elementi meno uno. Si assumerà che il puntatore PTR1 punti al primo elemento del Blocco 1, PTR2 al primo elemento del Blocco 2 e PTR3

punti all'area di destinazione dove dovrebbe essere immagazzinato il risultato.

Il programma è autoesplicativo. L'ultimo elemento del Blocco 1 è letto nell'accumulatore e quindi sommato all'ultimo elemento del Blocco 2. Esso è immagazzinato alla locazione appropriata del Blocco 3. L'elemento sequenzialmente successivo viene sommato e così via.

### Alcuni esercizi utilizzanti l'Indirizzamento Indiretto Indicizzato

Qui si assuma che gli indirizzi PTR1, PTR2, PTR3 non siano inizialmente noti. Comunque si conosce che essi sono immagazzinati in pagina 0 agli indirizzi LOC 1, LOC 2, LOC 3.
Questo è un meccanismo comune per il passaggio delle informazioni tra subroutine. Il programma corrispondente appare di seguito:

```
BLKADD  LDY   # NBR — 1
NEXT    CLC
        LDA   (LOC1), Y
        ADC   (LOC2), Y
        STA   (LOC3), Y
        DEY
        BPL   NEXT
```

La corrispondenza tra questo nuovo programma ed il precedente potrebbe non essere ovvia. Esso illustra chiaramente l'uso del meccanismo indiretto indicizzato ogni volta che l'indirizzo assoluto non è noto all'istante in cui viene scritto il programma, ma è nota la locazione dell'informazione. Si può notare che i due programmi hanno esattamente lo stesso numero di istruzioni. Un interessante esercizio è ora la determinazione di quale sarà eseguito più velocemente.

**Esercizio 5.2:** *Si calcoli il numero di byte ed il numero di cicli per ciascuno di questi due programmi, utilizzando le tabelle riportate nella sezione delle appendici.*

## SOMMARIO

È stata presentata una descrizione completa dei modi di indirizzamento. È stato mostrato che il 6502 offre la maggior parte dei meccanismi possibili e sono state analizzate le sue caratteristiche. Infine sono stati presentati alcuni programmi di applicazione per dimostrare il valore dei meccanismi di indirizzamento. La programmazione del 6502 richiede la comprensione di questi meccanismi.

## ESERCIZI

**Esercizio 5.3:** *Si scriva un programma per sommare i primi 10 byte di una tabella immagazzinata alla locazione "BASE". Il risultato avrà 16 bit. (Questo è un calcolo di tipo checksum).*

**Esercizio 5.4:** *Si può risolvere lo stesso problema senza utilizzare il modo di indicizzazione.*

**Esercizio 5.5:** *Si inverta l'ordine dei 10 byte di questa tabella. Si immagazzini il risultato all'indirizzo "REVER".*

**Esercizio 5.6:** *Si cerchi l'elemento più grande della stessa tabella. Lo si immagazzini all'indirizzo di memoria "LARGE".*

**Esercizio 5.7:** *Si sommino insieme gli elementi corrispondenti di tre tabelle le cui basi sono BASE1, BASE2, BASE3. La lunghezza di queste tabelle è immagazzinata in pagina zero all'indirizzo "LENGTH".*

CAPITOLO 6

# TECNICHE D'INGRESSO/USCITA

## INTRODUZIONE

Si è imparato come scambiare l'informazione tra la memoria ed i vari registri del processore. Si è imparato a dirigere i registri e ad utilizzare una quantità di istruzioni per manipolare i dati. Si deve imparare ora a comunicare i dati col mondo esterno. Questo è chiamato ingresso/u-scita.

*L'ingresso* fa riferimento alla cattura di dati dalle periferiche esterne (tastiera, disk oppure sensore fisico).

*L'uscita* fa riferimento al trasferimento di dati dal microprocessore o dalla memoria ai dispositivi esterni come una stampante, un CRT, un disk oppure relé o sensori effettivi.

Si procederà in due fasi. Prima si imparerà ad eseguire le operazioni d'ingresso/uscita richieste dai dispositivi comuni. In seguito si imparerà a dirigere diversi dispositivi d'ingresso/uscita contemporaneamente o scheduling. Questa seconda parte coprirà in particolare la scelta in funzione degli interrup.

## INGRESSO/USCITA

In questo paragrafo si imparerà a rivelare od a generare segnali semplici come impulsi. Quindi si studieranno le tecniche per imporre o misurare un timing corretto. Si sarà quindi pronti per tipi più complessi di ingresso/uscita come i trasferimenti seriale o parallelo ad alta velo-cità.

### Generazione di un Segnale

Nel caso più semplice un dispositivo d'uscita sarà spento (o acceso) dal calcolatore. Per cambiare lo stato del dispositivo d'uscita, il programma-tore cambierà semplicemente un livello da uno "0" logico ad un "1" logico oppure da "1" a "0". Si assumerà che un relé esterno sia connesso al bit "0" di un registro chiamato "OUT 1". Per eccitarlo si scriverà semplicemente un "1" nella posizione di bit appropriata del registro. Qui

si assumerà che OUT1 rappresenti l'indirizzo di questo registro d'uscita all'interno del sistema. Il programma che eccita il relé è:

```
TURNON      LDA        # % 00000001
            STA        OUT1
```

È stato assunto che lo stato degli altri sette bit del registro OUT1 siano trascurabili. Comunque spesso non è così. Questi bit devono essere connessi ad altri relé. Si migliorerà perciò questo programma semplice. Si vuole commutare l'eccitazione del relé senza cambiare lo stato di qualsiasi altro bit all'interno di questo registro. Si assumerà che sia possibile leggere o scrivere i contenuti di questo registro. Il programma migliorato diviene:

```
TURNON      LDA   OUT1            LEGGE I CONTENUTI DI OUT1
            ORA   # % 00000001    FORZA AD "1" IL BIT 0
                  STA OUT1
```

Questo programma legge prima i contenuti della locazione OUT1 quindi esegue un OR inclusivo dei suoi contenuti. Questo cambia solo ad "1" la posizione di bit 0 e lascia intatto il resto del registro. (Per ulteriori dettagli sull'operazione ORA si faccia riferimento al Capitolo 4). Questo è illustrato dalla Figura 6-1.

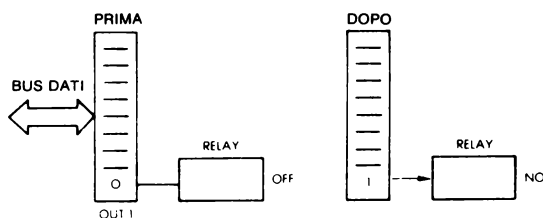



Figura 6.1: Eccitazione di un relè

## Impulsi

La generazione di un impulso è eseguita esattamente come nel caso del *livello* precedente. Un bit di uscita è prima commutato on e successivamente commutato off. Questo origina un impulso. Questo è illustrato in Figura 6-2. Per quanto riguarda questo tempo occorre risolvere un problema aggiuntivo: si deve generare l'impulso per la lunghezza di tempo corretta. Si studierà perciò la generazione di un ritardo calcolato.

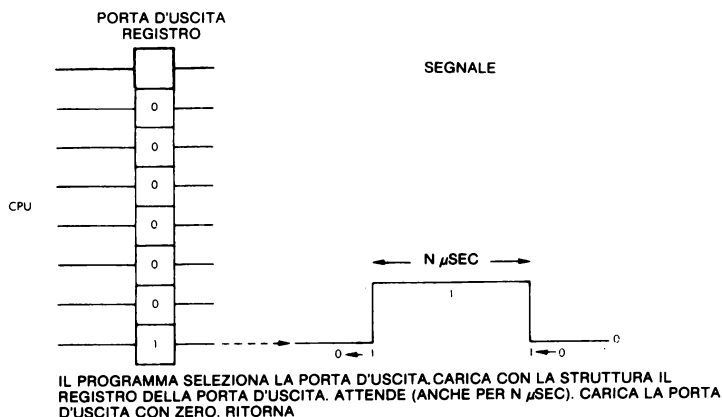



Figura 6.2: Un impulso programmato

## Generazione e Misura di Ritardo

Un ritardo può essere generato mediante metodi software oppure hardware. Si studierà qui il modo per eseguirlo mediante un programma e successivamente si mostrerà come esso possa essere realizzato con un contatore hardware, detto temporizzatore ad intervallo programmabile (PIT).

I ritardi programmati sono ottenuti mediante conteggio. Un registro contatore è caricato con un certo valore e quindi decrementato. Il programma cicla su sè stesso e si decrementa finchè il contatore raggiunge il valore "0". La lunghezza totale di tempo utilizzata da questo processo realizzerà il ritardo richiesto. Come esempio si genererà un ritardo di 37 microsecondi.

```
DELAY   LDY   # 07    Y È IL CONTATORE
NEXT    DEY           DECREMENTA
        BNE   NEXT    TEST
```

Questo programma carica il registro indice Y col valore 7. L'istruzione successiva decrementa Y e l'ulteriore istruzione successiva causa una diramazione a NEXT finchè Y non è decrementato a "0". Quando infine Y è decrementato a "0" il programma uscirà da questo ciclo ed eseguirà qualunque istruzione successiva. La logica del programma è semplice ed appare nel diagramma di flusso della Figura 6-3.

Si calcolerà ora il ritardo effettivo realizzato dal programma. Osservando il paragrafo di appendice del libro si troverà il numero di cicli richiesto da ciascuna di queste istruzioni.

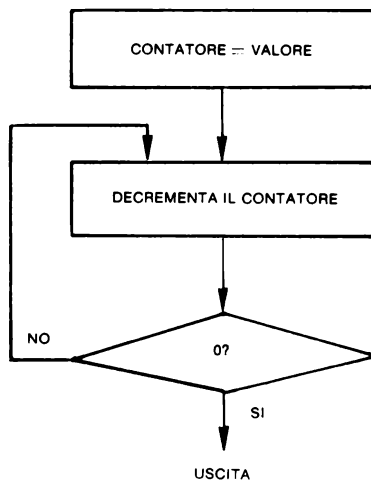



Figura 6.3: Diagramma di flusso di un ritardo

LDY, nel modo immediato, richiede 2 cicli. DEY utilizzerà 2 cicli. Osservando il numero di cicli nella tabella per BNE si verifica la diramazione BNE richiederà solo 2 cicli. Se si verifica la diramazione, che sarà il caso normale durante il ciclo, è richiesto un ulteriore ciclo. Infine se si deve attraversare il confine della pagina allora è richiesto un ciclo ulteriore. Qui si assume che non si debba attraversare la frontiera della pagina.

Il timing è perciò 2 cicli per la prima istruzione, più 5 cicli per le 2 successive moltiplicazioni per il numero di volte di esecuzione del ciclo:
Ritardo $= 2 + 5 \times 7 = 37$

Assumendo un tempo di ciclo di 1 microsecondo questo ritardo programmato sarà di 37 microsecondi.

Si può vedere da questo esempio semplice che la massima definizione con cui si può regolare la lunghezza del ritardo è 2 microsecondi. Il ritardo minimo è 2 microsecondi.

**Esercizio 6-1:** *Qual'è il massimo ritardo che può essere realizzato con queste tre istruzioni?*

**Esercizio 6-2:** *Si modifichi il programma per ottenere un ritardo di circa 100 microsecondi.*

Se si desidera realizzare un ritardo più lungo, una soluzione semplice è di aggiungere ulteriori istruzioni nel programma, tra DEY e BNE. Il modo più semplice per fare questo è di aggiungere istruzioni NOP (l'istruzione NOP non opera per 2 cicli).

## Ritardi più lunghi

La generazione di ritardi più lunghi mediante software può essere ottenuta utilizzando un contatore più largo. Due registri interni, o meglio due parole della memoria, possono essere utilizzati per conservare un conteggio a 16 bit. Per semplicità si assuma che il conteggio più basso sia "0". Il byte più basso sarà caricato con "255", il conteggio massimo e quindi si entra nel ciclo che lo decrementa. Ogni volta che esso è decrementato a "0" il byte superiore del conteggio sarà decrementato al valore "0" il programma termina. Se è richiesta più precisione nella generazione del ritardo, il conteggio più basso può avere un valore non zero. In questo caso si scriverebbe il programma come spiegato e si aggiungerebbe alla fine il programma di tre righe di generazione del ritardo che è stato appena descritto.

Naturalmente ritardati ancora più lunghi possono essere generati utilizzando più di due parole. Questo è analogo al modo in cui opera un contachilometri su una automobile.
Quando la ruota all'estrema destra va da "9" a "0" la ruota che la precede a sinistra viene incrementata di 1. Questo è il principio generale del conteggio con unità discrete multiple

Comunque l'obiezione principale è che conteggiando ritardi il microprocessore non farà nient'altro per centinaia di millisecondi od anche secondi. Se il computer non ha nient'altro da fare è perfettamente accettabile. Comunque, nel caso generale, il microprocessore dovrebbe essere disponibile per altri compiti cosicchè i ritardi più lunghi non sono normalmente realizzati mediante software. Infatti anche i ritardi più corti possono essere accettabili in un sistema se questo deve fornire risposte in tempo garantito in assegnate situazioni. Occorre utilizzare i ritardi hardware. Inoltre se si utilizzano gli interrupt, la precisione del timing può andare perduta se il ciclo di conteggio è interrotto.

**Esercizio 6-3:** *Si scriva un programma per realizzare un ritardo di 100 ms (per una telescrivente).*

### Ritardi Hardware

I ritardi hardware sono realizzati utilizzando un temporizzatore di ritardo automatico o brevemente "temporizzatore". Un registro del temporizzatore viene caricato con un valore. La differenza è che questa volta il temporizzatore decrementerà automaticamente e periodicamente questo contatore. Il periodo è normalmente regolabile o selezionabile dal programmatore. Ogni volta che il temporizzatore sarà decrementato a "0" esso invierà normalmente un interrupt al microprocessore. Esso porrà anche un bit di stato che può essere rivelato periodicamente dal contatore. L'impiego degli interrupt sarà spiegato successivamente in questo capitolo.

Altri modi di funzionamento del temporizzatore possono comprendere la partenza da "0" ed il conteggio della durata del numero di impulsi ricevuti. Quando sta funzionando come un temporizzatore ad un intervallo si dice che funziona in un modo *one-shot.* Quando sta contando impulsi si dice che funziona in un modo a *conteggio d'impulso.* Alcuni dispositivi temporizzatori possono anche comprendere registri multipli ed un certo numero di possibilità a scelta che sono preselezionate dal programma. Questo è il caso, per esempio, dei temporizzatori contenuti nel componente 6522, un chip I/O che sarà descritto al capitolo successivo.

### Rivelazione di impulsi

La rivelazione di impulsi è il problema inverso della generazione di impulsi con in più un'ulteriore difficoltà: mentre un impulso di uscita è generato sotto il controllo del programma, l'impulso d'ingresso si verifica in modo *asincrono* col programma. Per rivelare un impulso si possono utilizzare due metodi: *registrazione ed interrupt.* Gli interrupt saranno descritti in seguito in questo capitolo. Si consideri ora la tecnica di registrazione. Utilizzando questa tecnica il programma legge il valore di un dato registro d'ingresso in modo continuo, verificando una posizione di bit, forse il bit 0. Si assumerà che il bit 0 sia originariamente "0". Ogni volta che viene ricevuto un impulso questo bit assumerà il valore "1". Il programma osserva continuamente il bit 0 finchè esso assume il valore "1". Quando si trova un "1", l'impulso è stato rivelato. Il programma è il seguente:

```
POLL        LDA      # $01
AGAIN       BIT      INPUT
            BPL      AGAIN
ON
```

Inversamente si assuma che la linea d'ingresso sia normalmente "1" e che si voglia rivelare uno "0". Questo è il caso normale di rivelazione del bit START, quando si sta osservando una linea connessa ad una telescrivente. Il programma è il seguente:

```
POLL      LDA    # $01
NEXT      BIT    INPUT
          BMI    NEXT
START
```

### Controllo della Durata

Il controllo della durata dell'impulso può essere realizzata allo stesso modo del calcolo della durata di un impulso di uscita. Si può utilizzare una tecnica hardware oppure software. Quando si sta controllando un impulso mediante software un contatore è regolarmente incrementato di 1 quando è verificata la presenza dell'impulso. Se l'impulso è ancora presente il programma cicla ancora su sè sesso. Ogni volta che l'impulso scompare, il conteggio contenuto nel registro contatore è utilizzato per calcolare là durata effettiva dell'impulso. Il programma è il seguente:

```
DURTN     LDX    # 0       AZZERA IL CONTATORE
          LDA    # $01     CONTROLLO BIT 0
AGAIN     BIT    INPUT
          BPL    AGAIN
LONGER    INX
          BIT    INPUT
          BMI    LONGER
```

Naturalmente si assumerà che la massima durata dell'impulso non origini l'overflow del registro X. Se succedesse questo il programma dovrebbe essere più lungo per tener conto di questo (oppure questo potrebbe essere un errore programmato!).

Poichè ora si conosce come rivelare e generare gli impulsi si consideri il trasferimento di grandi quantità di dati. Si distingueranno due casi: dati seriali e dati paralleli. Quindi si applicherà questo ai dispositivi d'ingresso/uscita effettivi.

## TRASFERIMENTO PARALLELO DI PAROLA

Qui si assume che gli otto bit dei dati del trasferimento siano disponibili in parallelo all'indirizzo "INPUT". Il microprocessore deve leggere la parola dei dati in questa locazione ogni volta che una parola di stato indica che essa è valida. L'informazione di stato sarà assunta contenuta

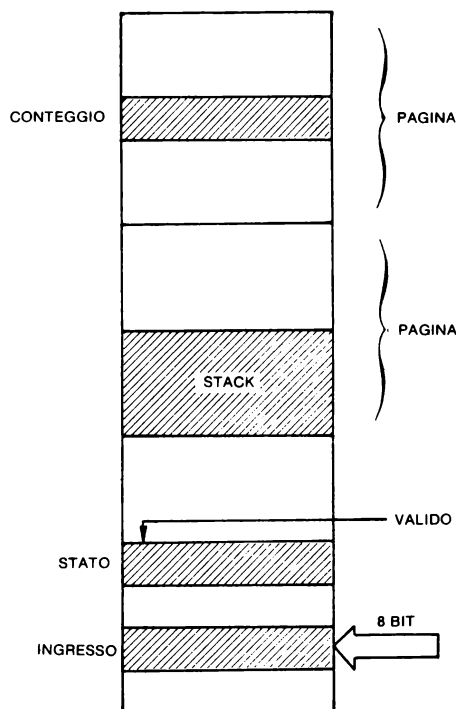



Figura 6.4: Trasferimento parallelo di parola: la memoria

nel bit 7 dell'indirizzo "STATUS". Qui si scriverà un programma che leggerà e conserverà automaticamente ogni parola dei dati entranti. Per semplicità si assumerà che il numero di parole da leggere sia inizialmente noto e sia contenuto nella locazione "COUNT". Se quest'informazione non fosse disponibile si dovrebbe verificare il cosidetto *carattere di rottura,* come una *cancellazione,* oppure il carattere "*". Si è già imparato a fare questo.

Il diagramma di flusso appare in Figura 6-5. È abbastanza diretto. Si verifica l'informazione di stato finchè essa diviene "1" indicando che una parola è pronta. Quando la parola è pronta viene letta e conservata in un'appropriata locazione di memoria. Si decrementa quindi il contatore e si verifica se esso è stato decrementato a "0". In questo caso si è terminato; altrimenti si legge la parola successiva. Il programma che realizza questo algoritmo è il seguente:

```
PARAL   LDX   COUNT    CONTATORE
WATCH   LDA   STATUS   IL BIT 7 È 1 SE IL DATO È VALIDO
        BPL   WATCH    DATO VALIDO?
        LDA   INPUT    LO LEGGE
        PHA            LO CONSERVA NELLO STACK
        DEX
        BNE   WATCH
```

Le prime due istruzioni del programma leggono l'informazione di stato e causano l'instaurarsi di un ciclo non appena il bit 7 del registro di stato è "0". (Esso è il bit segno cioè il bit N).

```
WATCH    LDA  STATUS
         BPL  WATCH
```

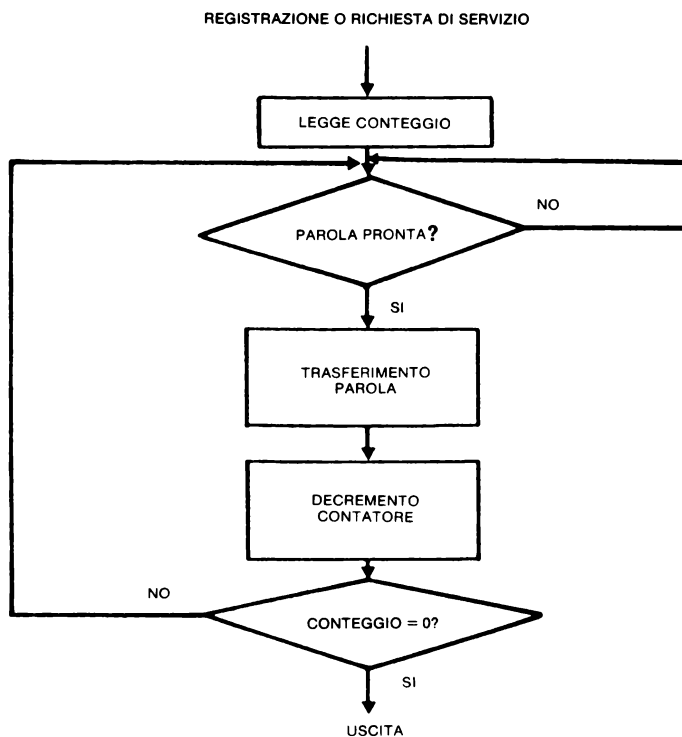



Figura 6.5: Trasferimento parallelo di parola: diagramma di flusso

Quando BPL non è soddisfatta il dato è valido e si può leggerlo:

```
LDA INPUT
```

La parola che è stata letta dall'indirizzo INPUT dove si trova, deve essere conservata. Assumendo che il numero di parole da trasmettere sia abbastanza piccolo si utilizza:

```
PHA
```

Se lo stack fosse pieno ovvero fosse grande il numero di parole da trasferire non si potrebbe spingerlo nello stack e si dovrebbe trasferirlo ad un'assegnata area di memoria utilizzando, per esempio, un'istruzione indicizzata. Comunque questo richiederebbe un'ulteriore istruzione per incrementare o decrementare il registro indice. PHA è più veloce.

La parola del dato è quindi stata letta e conservata. Si decrementerà semplicemente il contatore di parole e si verificherà se si è finito:

```
DEX
BNE WATCH
```

Si rimarrà nel ciclo finchè il contatore eventualmente decrementa a "0". Questo programma di 6 istruzioni può essere chiamato un banco di prova. Un programma *banco di prova* è un programma attentamente ottimizzato progettato per verificare le possibilità di un dato processore in una situazione specifica. I trasferimenti paralleli sono una di tali situazioni tipiche. Questo programma è stato progettato per una massima velocità ed efficienza. Si calcolerà ora la massima velocità di trasferimento di questo programma. Si assumerà che COUNT sia contenuto in pagina 0. La durata di ogni istruzione è determinata dall'ispezione della tabella alla fine del libro e si trova essere la seguente:

```
                                  CICLI
          LDX     COUNT             3
WATCH     LDA     STATUS            4
          BPL     WATCH            2/3     (INSODDISFATTO/
                                            SODDISFATTO)
          LDA     INPUT             4
          PHA                       3
          DEX                       2
          BNE     WATCH            2/3     (INSODDISFATTO/
                                            SODDISFATTO)
```

Il tempo minimo di esecuzione è ottenuto assumendo che il dato sia disponibile ogni volta che si campiona STATUS. In altre parole la prima

BPL non sarà soddisfatta tutte le volte. Il timing è quindi: $3 + (4 + 2 + 4 + 3 + 2 + 3) \times COUNT$.

Trascurando i primi 3 microsecondi necessari per inizializzare il registro contatore, il tempo impiegato per trasferire una parola è 18 microsecondi.

La massima velocità di trasferimento è perciò:

$$\frac{1}{18\,(10^{-6})} = 55 \text{ K byte al secondo.}$$

**Esercizio 6-4:** *Si assuma che il numero di parole da trasferire sia maggiore di 256. Si modifichi il programma di conseguenza e si determini l'influenza sulla massima velocità di trasferimento.*

Si è vista l'esecuzione di trasferimenti paralleli ad alta velocità. Di seguito si considera un caso più complesso.

## TRASFERIMENTO SERIALE DI BIT

Un ingresso è seriale se i bit dell'informazione (zeri ed uni) entrano successivamente su una linea. Questi bit possono entrare ad intervalli regolari. Questa è chiamata normalmente trasmissione *sincrona.* Oppure essi possono entrare come raffica di dati ad intervalli casuali. Questa è chiamata trasmissione *asincrona.* Si svilupperà un programma che possa lavorare in entrambi i casi. Il principio della cattura sequenziale di dati è semplice: si osserverà una linea d'ingresso che sarà assunta essere la linea 0. Quando un bit dei dati sarà rivelato su questa linea si leggerà il bit di ingresso e lo si sposterà in un registro per conservarlo. Ogni volta che si sono accumulati 8 bit si preserverà il byte di dati nella memoria e si costruisce quello successivo. Per semplicità si assumerà che il numero di byte da ricevere sia inizialmente noto. Diversamente occorre, per esempio, osservare uno speciale carattere di interruzione ed arrestare il trasferimento seriale di bit a questo punto. Si è già imparato a fare questo. Il diagramma di flusso è riportato in Figura 6-7. Il programma è il seguente:

```
SERIALE  LDA  # $00
         STA  WORD
LOOP     LDA  INPUT    IL BIT 7 È LO STATO, "0" È IL DATO
         BPL  LOOP     RICEVUTO IL BIT?
         LSR  A        LO SPOSTA IN C
         ROL  WORD     CONSERVA IL BIT IN MEMORIA
         BCC  LOOP     CONTINUA SE CARRY = "0"
         LDA  WORD
         PHA           CONSERVA IL BYTE ASSEMBLATO
```

```
LDA   # $01    RIPRISTINA IL CONTATORE DI BIT
STA   WORD
DEC   COUNT    DECREMENTA IL CONTEGGIO DI PAROLA
BNE   LOOP     ASSEMBLA LA PAROLA SUCCESSIVA
```

Questo programma è stato progettato per un'alta efficienza e si utilizza una nuova tecnica che si spiegherà. (Vedere Figura 6-6).

Le convenzioni sono le seguenti: si assume che la locazione di memoria COUNT contenga un conteggio del numero di parole da trasferire. La locazione WORD sarà utilizzata per assemblare 8 bit entranti consecutivi. L'indirizzo INPUT fa riferimeno ad un registro d'ingresso. Si

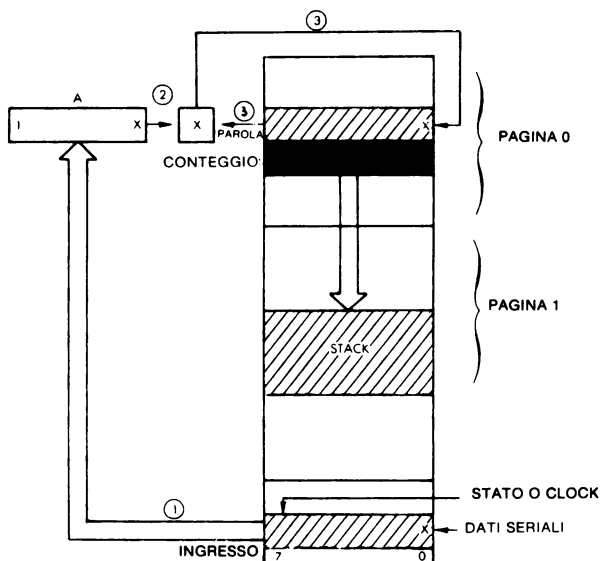



Figura 6.6: Conversione da seriale a parallelo

assuma che la posizione di bit 7 di questo registro sia un flag di stato oppure un bit di clock. Quando esso è "0" il dato non è valido. Quando esso è "1" il dato ę valido. Il dato stesso sarà assunto apparire nella posizione di bit 0 di questo stesso indirizzo. In molti casi l'informazione di stato apparirà su un registro diverso dal registro dati.

Sarebbe quindi abbastanza semplice modificare conseguentemente questo programma. Inoltre si assumerà che il primo bit dei dati che questo programma riceve sia garantito essere un "1". Questo indica che

segue il dato effettivo. Se non fosse così si considererà successivamente una ovvia modifica. Il programma corrisponde esattamente al diagramma di flusso della Figura 6-7. Le primissime righe del programma realizzano un ciclo di attesa che verifica se un bit è pronto. Per determinare se un bit è pronto si legge il registro d'ingresso che verifica il bit segno (N). Finchè questo bit è "0" l'istruzione BPL è soddisfatta e si avrà la diramazione di ritorno del ciclo. Ogni volta che il bit di stato (oppure il clock) diverrà vero ("1") BPL sarà insoddisfatta e sarà eseguita l'istruzione sequenzialmente successiva.

Si ricordi che BPL significa "opera la diramazione se Positivo", cioè quando il bit 7 (il bit segno) è "0". La sequenza iniziale di istruzioni corrisponde alla freccia 1 in Figura 6-6.

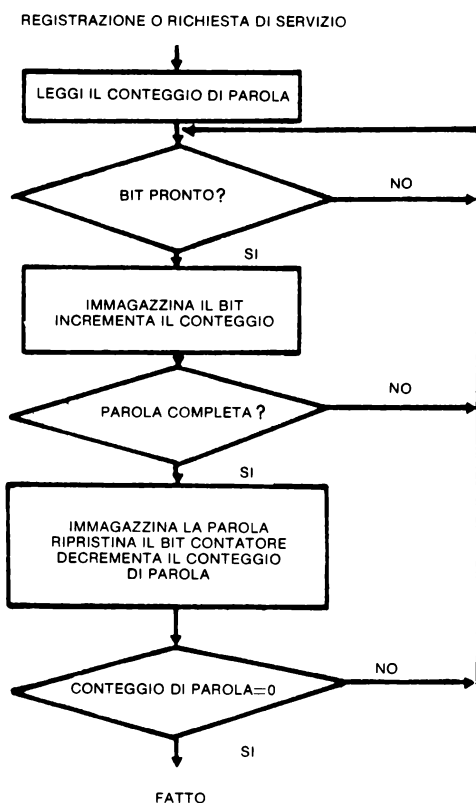



Figura 6.7: Trasferimento seriale di bit: diagramma di flusso

A questo punto l'accumulatore contiene un "1" nella posizione di bit 7 ed il dato effettivo nella posizione di bit 0. Il primo bit dati che arriva deve essere un "1". Comunque quelli successivi possono essere sia "0" che "1". Si desidera ora preservare il bit dato collocato in posizione 0. L'istruzione:

LSR A

fa scorrere i contenuti dell'accumulatore a destra di una posizione. Questo fa cadere il bit più a destra di A, che è il bit dato, nel bit carry. Si preserverà ora questo bit dato nella locazione di memoria WORD: (questo è illustrato dalle frecce 2 e 3 nella Figura 6-6).

ROL WORD

L'effetto di questa istruzione è la lettura del bit carry nella posizione di bit più a destra dell'indirizzo WORD. Nello stesso tempo il bit più a sinistra di WORD cade nel bit carry. (Se si ha qualche dubbio sull'operazione di rotazione,si faccia riferimento al Capitolo 4).

È importante ricordare che un'operazione di rotazione salverà il bit carry, qui nella posizione estrema destra, ed anche il ripristino del bit carry col valore del bit 7.

Qui uno "0" cadrà nel carry. L'istruzione successiva:

BCC LOOP

verifica il carry ed opera la diramazione indietro all'indirizzo LOOP finchè il carry è "0". Questo è il contatore di bit automatico. Si può immediatamente vedere che, come risultato della prima ROL, WORD conterrà "00000001". Otto scorrimenti dopo l'"1" cadrà finalmente nel bit carry e si arresterà la diramazione. Questo è un modo ingegnoso per realizzare un contatore di ciclo automatico senza dover sprecare un'istruzione per decrementare i contenuti di un registro indice. Questa tecnica è utilizzata per abbreviare il programma e migliorare le sue caratteristiche.

Ogni volta che BCC infine non è soddisfatta, 8 bit sono stati assemblati nella locazione WORD. Questo valore dovrebbe essere preservato nella memoria. Questo è realizzato dalle istruzioni successive:

LDA WORD
PHA

Qui si stanno conservando i dati di WORD (8 bit) nello stack. La conservazione nello stack è possibile solo se è disponibile lo spazio sufficiente. Supponendo che questa condizione sia soddisfatta questo è il

modo più veloce per preservare una parola nella memoria. Il puntatore dello stack viene aggiornato automaticamente. Se non si ponesse una parola nello stack si dovrebbe utilizzare un'ulteriore istruzione per aggiornare un puntatore della memoria. Si potrebbe equivalentemente eseguire un indirizzamento indicizzato ma questo comprenderebbe l'incremento ed il decremento dell'indice, utilizzando un tempo ulteriore.

Dopo che la prima WORD di dati è stata conservata non si ha nessuna garanzia che il primo bit dei dati che entreranno sarà un "1". Potrebbe essere qualsiasi. Si deve perciò ripristinare i contenuti di WORD a "00000001" così da poterla utilizzare come un bit contatore. Questo è eseguito dalle due istruzioni successive:

```
LDA # $01
STA  WORD
```

Infine si decrementerà il contatore di parola poichè una parola è stata assemblata e si verificherà se si è raggiunta la fine del trasferimento. Questo è eseguito dalle due istruzioni successive:

```
DEC COUNT
BNE LOOP
```

Il programma precedente è stato progettato per alta velocità cosicchè esso possa catturare una corrente d'ingresso veloce di bit dati. Una volta che il programma termina occorre naturalmente leggere immediatamente dallo stack le parole ivi conservate e trasferirle dovunque nella memoria. Si è già imparato ad eseguire un tale trasferimento di blocco nel Capitolo 2.

**Esercizio 6-5:** *Si calcoli la velocità massima con cui questo programma sarà in grado di leggere i bit seriali. Per calcolare questa velocità si assuma che gli indirizzi WORD e COUNT siano mantenuti in Pagina 0. Si assuma inoltre che il progamma completo risieda all'interno della stessa pagina. Si consulti il numero di cicli richiesto da ciascuna istruzione nella tabella alla fine di questo libro e quindi si calcoli il tempo che trascorrerà durante l'esecuzione di questo programma. Per calcolare la lunghezza del tempo utilizzato da un ciclo, espressa in microsecondi, per il numero di volte che esso sarà eseguito. Inoltre nel calcolo della massima velocità si assuma che un dato sia pronto ogni volta che viene rivelata la locazione d'ingresso.*

Questo programma è molto più difficile da capire rispetto ai precedenti. Lo si osservi ancora (riferimento alla Figura 6-6) in dettagli ulteriori esaminando alcuni compromessi.

Un bit dei dati entra nella posizione di bit 0 di "INPUT" di volta in

volta. Potrebbero esserci per esempio tre "1" in successione. Si deve perciò differenziare tra i bit entranti successivi. Questa è la funzione del segnale di clock.

Il segnale di clock (dello STATUS) dice che il bit d'ingresso è ora valido. Se lo stato è "0" si deve attendere. Se esso è "1" allora il bit dati è valido.

Si assumerà qui che il segnale di stato sia connesso al bit 7 del registro INPUT.

**Esercizio 6-6:** *Si saprebbe spiegare perchè il bit 7 è utilizzato per lo stato ed il bit 0 per i dati?*

Una volta che si è catturato un bit dati si deve preservarlo in una locazione sicura e quindi farlo scorrere a sinistra cosicchè si possa prendere il bit successivo.

Sfortunatamente l'accumulatore è utilizzato per leggere e verificare i dati e lo stato in questo programma. Se si vuole accumulare i dati nell'accumulatore, la posizione di bit 7 dovrebbe essere liberata dal bit di stato.

**Esercizio 6-7:** *Si saprebbe suggerire un modo per verificare lo stato senza liberare i contenuti dell'accumulatore (un'istruzione speciale)? Se questo può essere fatto, si potrebbe utilizzare l'accumulatore per accumulare i bit entranti successivi?*

**Esercizio 6-8:** *Si riscriva il programma utilizzando l'accumulatore per memorizzare i bit entranti. Lo si confronti col precedente in termini di velocità e numero di istruzioni.*

Si considerino due possibili variazioni:

È stato assunto che, nell'esempio particolare considerato, il primissimo bit entrante dovrebbe essere un carattere speciale, garantito essere un "1". Comunque nel caso generale esso può essere qualsisi.

**Esercizio 6-9:** *Si modifichi il programma precedente assumendo che il primissimo bit entrante sia un dato valido (da non scartare) e possa essere "0" ed "1". Suggerimento: il "contatore di bit" lavorerebbe correttamente se lo si inizializza col valore corretto.*

Infine è stata consevata la parola assemblata nello stack per guadagnare tempo. Si potrebbe naturalmente conservarla in una specificata area di memoria.

**Esercizio 6-10:** *Si modifichi il programma precedente e si conservi la parola WORD assemblata nell'area di memoria iniziando a BASE.*

**Esercizio 6-11:** *Si modifichi il programma precedente cosicchè il trasferimento si arresti quando il carattere "S" è rivelato nel flusso d'ingresso.*

### L'Alternativa Hardware

Come avviene di solito per molti algoritmi convenzionali d'ingresso/uscita, è possibile realizzare questa procedura mediante hardware. Il chip si chiama UART. Esso accumulerà automaticamente i bit. Comunque quando si desidera ridurre il conteggio di componenti questo programma, od una sua variazione, sarà conveniente utilizzarlo.

**Esercizio 6-12:** *Si modifichi il programma assumendo che i dati siano disponibili nella posizione di bit 0 della locazione INPUT mentre l'informazione di stato è disponibile nella posizione di bit 0 dell'indirizzo INPUT* + 1.

## SOMMARIO I/O DI BASE

Si è ora imparato ad eseguire operazioni d'ingresso/uscita elementari ed a dirigere un flusso di dati paralleli o di bit seriali. Si è ora pronti per comunicare con i dispositivi d'ingresso/uscita effettivi.

## COMUNICAZIONE CON I DISPOSITIVI I/O

Per scambiare dati con i dispositivi d'ingresso/uscita si dovrà innanzi tutto accertare se sono disponibili i dati, se si vuole leggerli oppure se il dispositivo è pronto ad accettare dati, se si vuole inviarglieli. Si possono usare due procedure: handshaking ed interrupt. Si studierà prima l'handshaking.

### Handshaking

L'handshaking è generalmente utilizzato nella comunicazione tra due dispositivi asincroni, cioè tra due dispositivi che non sono sincronizzati.

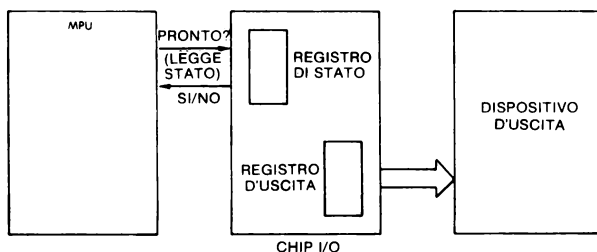



Figura 6.8: Handshaking (uscita)

Per esempio se si vuole inviare una parola ad una stampante parallela ci si deve prima assicurare che sia disponibile il buffer d'ingresso di questa stampante. Si chiederà perciò alla stampante: Sei Pronta? la stampante dirà "Si" oppure "No". Se essa non è pronta si attenderà. Se essa è pronta si invieranno i dati (Vedere Fig. 6-8).

Inversamente, prima della lettura di dati da un dispositivo d'ingresso si verificherà se i dati sono validi. Si chiederà: "Il dato è valido?". Ed il dispositivo dirà "Si" oppure "No". Il "Si" oppure "No" può essere indicato dai bit di stato, oppure da altri mezzi. (Vedere Figura 6-9).

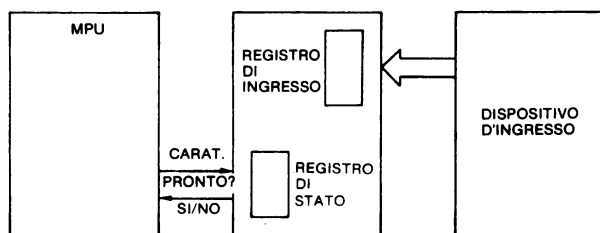



Figura 6.9: Handshaking (ingresso)

In breve ogni volta che si desidera scambiare informazioni con qualcuno che è indipendente e deve fare qualcos'altro si deve accertare che esso sia pronto alla comunicazione. La regola di cortesia usuale è di stringergli la mano e di qui segue il nome handshaking. Può quindi avvenire lo scambio di dati. Questa è la procedura normalmente utilizzata nella comunicazione con i dispositivi d'ingresso/uscita.

Si illustrerà ora questa procedura con un semplice esempio.

**Invio di un Carattere ad una Stampante**

Si assumerà che il carattere sia contenuto nella locazione di memoria CHAR. Il programma per stamparlo è il seguente:

```
CHARPR    LDX  CHAR      LEGGE IL CARATTERE
WAIT      LDA  STATUS    IL BIT 7 È "PRONTO"
          BPL  WAIT
          TXA
          STA  PRINT D
```

Il registro X viene prima caricato dalla memoria con il carattere da stampare. Quindi si verifica il bit di stato della stampante per determinare che essa sia pronta ad accettare il carattere. Comunque fino a che

essa non è pronta per stampare, si ha la diramazione all'indietro all'indirizzo WAIT e si cicla. Ogni volta che la stampante indica che essa è pronta a stampare ponendo ad "1" il suo bit pronto (qui convenzionalmente si è assunto il bit 7 dell'indirizzo STATUS) si può inviare il carattere. Si trasferisce il carattere dal registro X al registro A:

TXA

e lo si invia all'indirizzo del registro di uscita della stampante, qui indicato PRINTD.

STA PRINTD

**Esercizio 6-13:** *Si modifichi il programma precedente per stampare una stringa di n caratteri, dove n sarà assunto essere minore di 255.*

**Esercizio 6-14:** *Si modifichi il programma precedente per stampare una stringa di caratteri finchè non si incontra un codice di "ritorno carrello".*

Si complicherà ora la procedura di uscita richiedendo una conversione di codice e mediante alimentazione contemporanea di alcuni dispositivi:

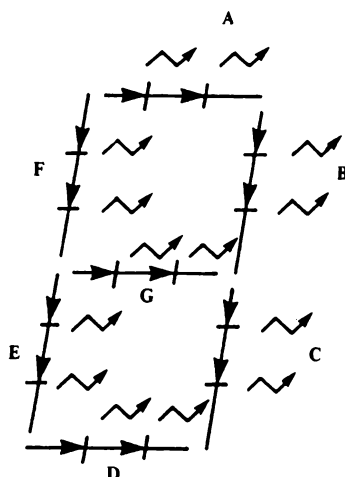



Figura 6.10: LED a sette segmenti

### Uscita su un LED a 7-Segmenti

Un tradizionale diodo-emettitore-di-luce (LED) a 7-segmenti può mostrare le cifre da "0" a "9" od anche i digit esadecimali da "0" ad "F" illuminando le combinazioni dei suoi 7 segmenti. Un LED a 7 segmenti è mostrato nell'illustrazione 6-10. I caratteri che possono essere generati

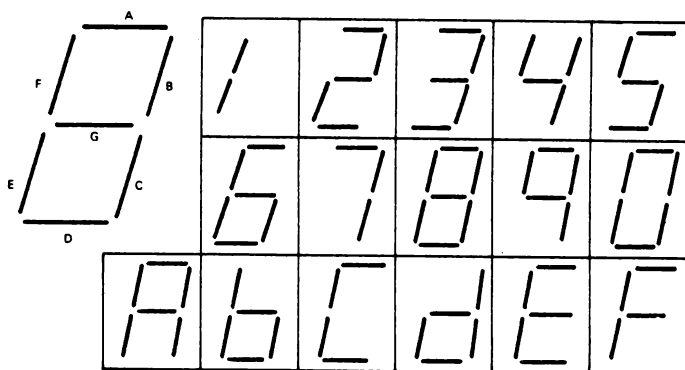



Figura 6.11: Caratteri generali con un LED a 7-segmenti

con questo LED appaiono in Figura 6-11.

I segmenti di un LED sono contrassegnati da "A"a "G" nella Figura 6-10. Per esempio "0" sarà mostrato illuminando i segmenti "ABC-DEF". Si assuma ora che il bit "0" di una porta d'uscita sia connesso al segmento "A", che "1" sia connesso al segmento "B" eccetera. Il bit 7 non viene utilizzato. Il codice binario richiesto per illuminare "FEDCBA" (per mostrare "0") è perciò "0111111". Questo in esadecimale è "3F". Si esegua l'esercizio seguente:

**Esercizio 6-15:** *Si calcoli l'equivalente a 7 segmenti dei digit esadecimali da "0" ad "F". Si riempia la tabella seguente:*

| Esa-dec. | Codice LED | Esa-dec. | Codice LED | Esa-dec. | Codice LED | Esa-dec. | Codice LED |
|---|---|---|---|---|---|---|---|
| 0 | 3F | 4 | | 8 | | C | |
| 1 | | 5 | | 9 | | D | |
| 2 | | 6 | | A | | E | |
| 3 | | 7 | | B | | F | |

Si mostreranno ora i valori esadecimali su LED *diversi.*

## Pilotaggio di LED multipli

Un LED non ha memoria. Esso mostrerà il dato solo finchè le sue linee segmento sono attive. Per mantenere basso il costo di un display LED il microprocessore mostrerà l'informazione *a turno in ciascuno dei LED.* La

rotazione tra i LED deve essere veloce sufficientemente da non provocare lampeggiamento apparente. Questo implica che il tempo consumato nel passagio da un LED al successivo sia minore di 100 millisecondi.
Si progetterà un programma che realizzi questo. Il registro Y sarà utilizzato per puntare il LED su cui si vuole mostrare un digit. Si assuma che l'accumulatore contenga il valore esadecimale da mostrare sul LED. Inizialmente occorre convertire il valore esadecimale nella sua rappresentazione a 7-segmenti. Al paragrafo precedente è stata costruita la tabella di equivalenza. Poichè si sta accedendo ad una tabella si utilizzerà il modo di indirizzamento indicizzato dove l'indice di spostamento sarà fornito dal valore esadecimale. Questo significa che il codice a 7 segmenti per il digit esadeciale # 3 è ottenuto osservando il terzo elemento della tabella a partire dalla base. L'indirizzo della base sarà chiamato SEGBAS. Il programma è il seguente:

```
LEDS   TAX                    UTILIZZA IL VALORE ESADECIMALE CO-
                              ME INDICE
       LDA    SEGBAS, X       LEGGE IL CODICE IN A
       LDX    # $00
       STX    SEGDAT          SPEGNE I DRIVER DEI SEGMENTI
       STA    SEGDAT          MOSTRA IL DIGIT
       LDX    #$70            QUALSIASI NUMERO GRANDE
       STY    SEGADR
DELAY  DEX
       BNE    DELAY
       DEY                    PUNTA AL LED SUCCESSIVO
       BNE    OUT
       LDY    LEDNBR
OUT    RTS
```

Il programma assume che il registro Y contenga il numero del LED che sarà successivamente illuminato e che il registro X contenga il digit da mostrare.

Il programma prima osserva il codice a 7 segmenti corrispondente al valore esadecimale contenuto nell'accumulatore con le sue prime due istruzioni. Le due istruzioni successive caricano "00" come il valore dei segmenti da mostrare, cioè li spegne. L'istruzione successiva seleziona quindi gli appropriati segmenti LED da mostrare : STY SEGADR.

Viene quindi realizzato un ciclo di ritardo di tre istruzioni prima della commutazione del LED successivo. Infine il puntatore LED viene decrementato. (Esso potrebbe anche essere incrementato).

Se il puntatore LED decrementa a "0" esso deve essere ricaricato con il numero LED più alto. Questo è eseguito dalle due istruzioni successive. Qui si è assunto che questa è una subroutine e quindi l'ultima istruzione è un RST: "ritorno da subroutine".

**Esercizio 6-16:** *Assumendo che il programma precedente sia una subroutine, si noterà che esso utilizza internamente i registri X ed Y e modifica i loro contenuti. Assumendo che la subroutine possa utilizzre liberamente l'area di memoria indicata dagli indirizzi T1, T2, T3, T4, T5, si aggiungano istruzioni all'inizio ed alla fine del programma in modo da garantire che, quando si ha il ritorno dalla subroutine, i contenuti dei registri X ed Y siano ancora gli stessi che si avevano all'inizio della subroutine.*

**Esercizio 6-17:** *Esercizio analogo al precedente ma si assuma che l'area di memoria T1, ecc. non sia disponibile per la subroutine. (Suggerimento: si ricordi che esiste un meccanismo incorporato in ogni calcolatore per preservare l'informazione in ordine cronologico).*

Sono stati così risolti alcuni problemi d'ingresso/uscita.
Si consideri il caso di una periferica effettiva: la telescrivente.

**Ingresso-Uscita di Telescrivente**

La telescrivente è un dispositivo seriale. Essa invia e riceve parole di informazione in un formato seriale. Ogni carattere è codificato in formato ASCII ad 8 bit (la tabella ASCII appare alla fine di questo libro). Inoltre ogni carattere è preceduto da un bit di "inizio" e termina con due bit di "stop". Nella cosiddetta interfaccia 20 mA current loop utilizzata molto frequentemente, lo stato della linea è normalmente ad "1". Questo è utilizzato per indicare al processore che la linea non è stata tagliata. L'inizio è una transizione da "0" ad "1'. Questo indica al dispositivo ricevente che seguono i bit dei dati. La telescrivente convenzionale è un dispositivo a 10 caratteri al secondo. Si è già stabilito che ogni carattere richiede 11 bit. Questo significa che la telescrivente trasmetterà 110 bit al secondo. Si può anche dire che è un dispositivo a 110 baud. Si progetterà un programma per fare uscire bit seriali della telescrivente alla velocità corretta.

Figura 6.12: Formato di una parola di telescrivente

Centodieci bit al secondo implica che i bit siano separati da 9,09 millisecondi. Questo dovrà essere la durata del ciclo di ritardo che sarà realizzato tra bit successivi. Il formato della parola di una telescrivente

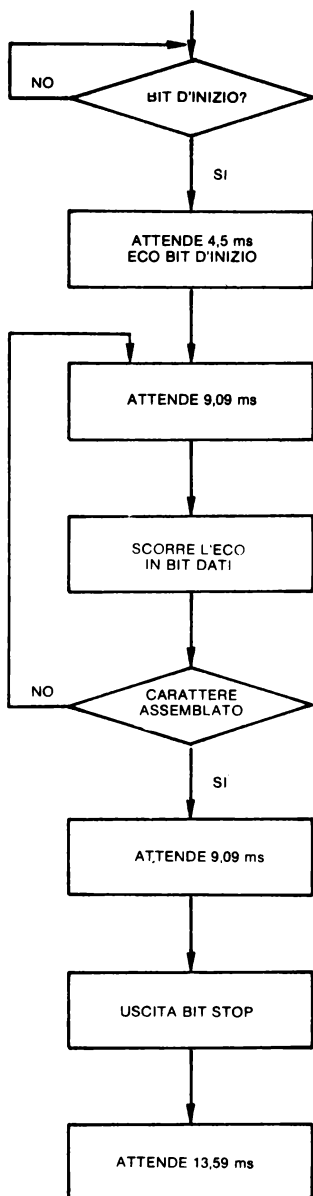



Figura 6.13: Ingresso con ECO alla telescrivente

appare in figura 6-13. Il programma è il seguente:

```
TTYN    LDA   STATUS
        BPL   TTYIN     REGISTRO USUALE DI STATO
        JSR   DELAY     ATTENDE 9,09 MS
        LDA   TTYBIT    BIT D'INIZIO
        STA   TTYBIT    RITORNO ECO
        JSR   DELAY
        LDX   # $08     BIT CONTATORE
NEXT    LDA   TTYBIT    SALVA L'INGRESSO
        STA   TTYBIT    RITORNO ECO
        LSR   A         CONSERVA IL BIT IN CARRY
        ROL   CHAR      CONSERVA IL BIT IN CHAR
        JSR   DELAY
        DEX             BIT SUCCESSIVO
        BNE   NEXT
        LDA   TTYBIT    STOP BIT
        STA   TTYBIT
        JSR   DELAY
        RTS
```

Figura 6.14: Ingresso da telescrivente

Si noti che questo programma differisce dal diagramma di flusso di Fig. 6-13.

Il programma dovrebbe essere esaminato con attenzione. La logica è abbastanza semplice. Il fatto nuovo è che se un bit è letto dalla telescrivente (all'indirizzo TTYBIT) esso rimanda l'eco alla telescrivente. Questa è una caratteristica convenzionale della telescrivente. Ogni volta che un utente preme un tasto l'informazione è trasmessa al processore e quindi ritorna al meccanismo stampante della telescrivente. Questo verifica che le linee di trasmissione sono operative e che il processore sta funzionando quando un carattere viene stampato correttamente su carta.

Le prime due istruzioni costituiscono il ciclo di attesa. Il programma attende che il bit di stato divenga vero ed inizia la lettura dei bit in ingresso. Come al solito il bit di stato è assunto entrare nella posizione di bit 7 poichè questa posizione può essere verificata in una sola istruzione da BPL (opera diramazione se positivo e questo è il bit segno).

JSR è il salto alla subroutine. Si utilizza una subroutine DELAY, per realizzare un ritardo di 9,09 ms. Si noti che DELAY può essere un ciclo di ritardo oppure può utilizzare un temporizzatore hardware se il sistema ne è dotato.

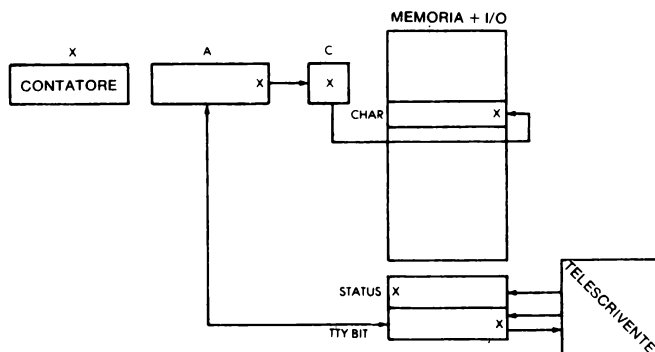



Figura 6.15: Ingresso telescrivente

Il primo bit ad entrare è il bit d'inizio. Deve arrivare l'eco alla telescrivente altrimenti viene ignorato. Questo viene fatto dalle istruzioni 4 e 5.

Ancora, si attende il bit successivo, ma questa volta è un bit dati vero e si deve conservarlo. Poichè tutte le istruzioni di scorrimento fanno cadere un bit nel flag carry, occorrono due istruzioni per preservare il bit dati. (L'X nella Figura 6-15): uno cade in C ("LSR A"), ed un altro per preservarlo nella locazione di memoria "CHAR" (ROL).

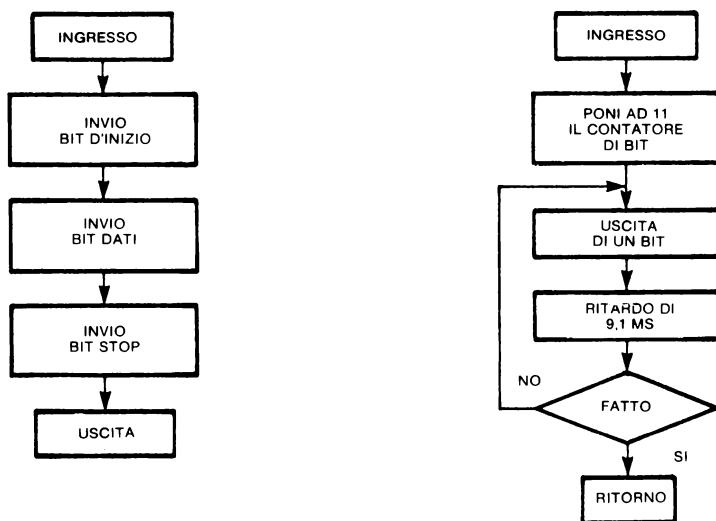



Figura 6.16: Uscita telescrivente

Attenzione ad un problema: l'istruzione "ROL" distruggerà i contenuti di C. Se si vuole che l'eco del bit dati ritorni occorre prendere la precauzione di preservarlo prima che esso scompaia in CHAR.

Infine si ha l'eco del bit dati: (STA TTY BIT) e si attende per quello successivo: (JSR DELAY) finchè si accumulano tutti gli otto bit dati: (DEX).

Ogni volta che si decrementa a zero, tutti gli 8 bit sono in CHAR. Rimane da ottenere l'eco dei bit STOP ed è finito.

**Esercizio 6-18:** *Si scriva la routine di ritardo che origina un ritardo di 9,09 millisecondi. (Subroutine DELAY).*

**Esercizio 6-19:** *Utilizzando l'esempio del programma precedente sviluppato si scriva un programma PRINTC che stampi su una telescrivente i contenuti della locazione di memoria CHAR.*

**Esercizio 6-20:** *Si modifichi il programma in modo che esso attenda un bit START invece di un bit STATUS.*

## Stampa di una Stringa di Caratteri

Si assumerà che la routine PRINTC (Vedere Esercizio 6-19) si occupi della stampa di un carattere su stampante, oppure display o qualsiasi dispositivo d'uscita. Qui si stamperanno i contenuti delle locazioni di memoria da (START + N) a (START).

Si utilizzerà naturalmente il modo di indirizzamento indicizzato ed il

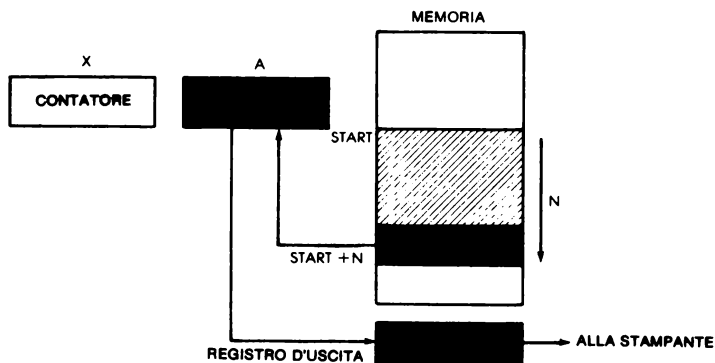



Figura 6.17: Stampa di un blocco di memoria

programma si ottiene direttamente:

```
PSTRING    LDX    # N          NUMERI DI PAROLE
NEXT       LDA    START + N
           JSR    PRINTC
           DEX
           BPL    NEXT
```

## SOMMARIO SULLE PERIFERICHE

Sono state descritte le tecniche di programmazione di base utilizzate per comunicare con dispositivi d'ingresso/uscita tipici. Inoltre per il

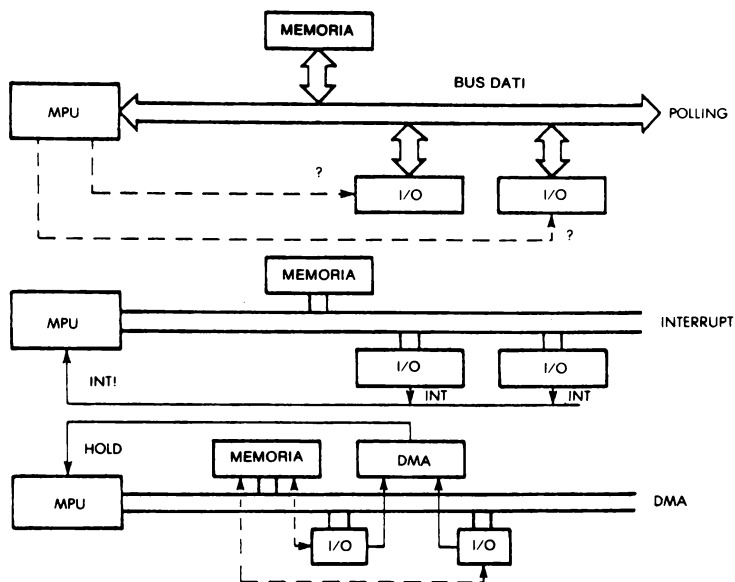



Figura 6.18: Tre metodi di controllo I/O

trasferimento di dati sarà necessario condizionare uno o più registri di controllo all'interno di ogni dispositivo I/O per condizionare correttamente le velocità di trasferimento, il meccanismo di interrupt e varie altre scelte. Si dovrebbe consultare il manuale di ciascun dispositivo. (Per maggiori dettagli sugli algoritmi specifici per scambiare l'informazione con tutte le periferiche più comuni si faccia riferimento al libro: "Tecniche di Interfacciamento per Microprocessori".

Si è ora imparato a dirigere dispositivi singoli. Comunque in un sistema reale tutte le periferiche sono connesse ai bus e possono richiedere contemporaneamente il servizio. Come si può eseguire lo scheduling del tempo del processore?

## SCHEDULING D'INGRESSO/USCITA

Poichè le richieste d'ingresso/uscita possono verificarsi contemporaneamente occorre realizzare in ogni sistema uno scheduling per determinare in quale ordine sarà concesso il servizio. Vengono utilizzate tre tecniche di base di ingresso/uscita che possono essere combinate con qualunque altra.
Essi sono: polling (registrazione), interrupt, DMA. Il polling e l'interrupt saranno descritte di seguito. Il DMA è una tecnica puramente hardware e come tale non sarà descritta qui.

### Registrazione (polling)

Concettualmente la registrazione è il metodo più semplice per la direzione di periferiche multiple. Con questa strategia il processore interroga i dispositivi connessi ai bus a turno. Se un dispositivo richiede

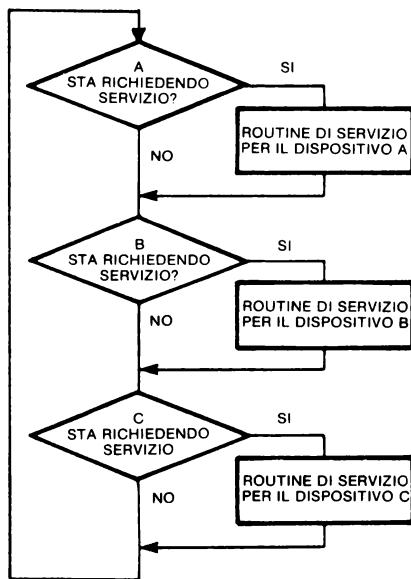



Figura 6.19: Diagramma di flusso del ciclo di registrazione

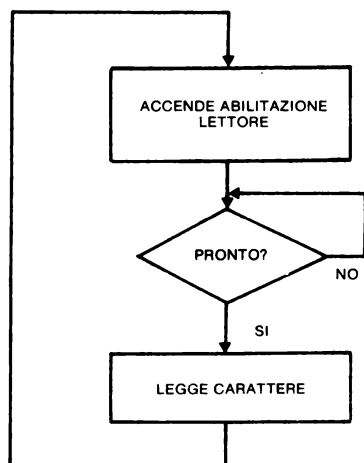



Figura 6.20: Lettura da un lettore di nastro di carta

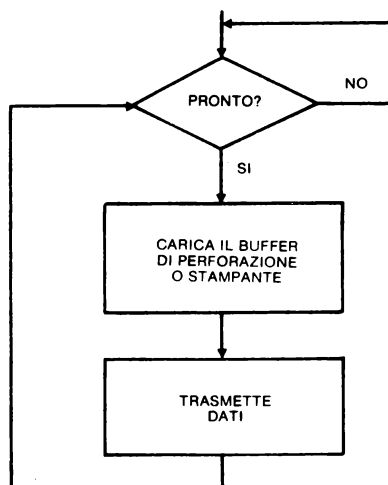



Figura 6.21: Stampa su una perforatrice o stampante

servizio questo viene concesso. Se non richiede servizio viene esaminata la periferica successiva. La registrazione non viene utilizzata per i dispositivi bensì per *qualsiasi routine di servizio del dispositivo.*

Per esempio, se un sistema è equipaggiato con una telescrivente, un registratore ed un display CRT, la routine di registrazione dovrebbe

interrogare la telescrivente: "hai un carattere da trasmettere?". Essa interrogherebbe la *routine di uscita* della telescrivente chiedendo "hai un carattere da inviare? Quindi, assumendo che la risposta sia negativa essa dovrebbe interrogare la routine del registratore nastro ed infine il display CRT.

Nel caso di un solo dispositivo connesso al sistema, la registrazione sarebbe utilizzata per determinare se è necessario il servizio. Come esempio nelle figure 6-20 e 6-21 appaiono i diagrammi di flusso per la lettura da un lettore di nastro di carta e la stampa su una stampante.

Esempio: un ciclo di registrazione per i dispositivi 1, 2, 3, 4, (vedere fig. 6-18):

```
POLL4   LDA   STATUS 1    LA RICHIESTA DI SERVIZIO
                          È IL BIT 7
        BMI   ONE
        LDA   STATUS 2    DISPOSITIVO 2?
        BMI   TWO
        LDA   STATUS 3    DISPOSITIVO 3?
        BMI   THREE
        LDA   STATUS 4    DISPOSITIVO 4?
        BMI   FOUR
        JMP   POLL 4      ALTRA VERIFICA
```

Il bit del registro di stato di ciascun dispositivo è "1" quando si richiede servizio. Quando è rivelata una richiesta questo programma opera una diramazione al dispositivo operatore, all'indirizzo ONE per il dispositivo 1, TWO per il dispositivo 2, ecc.

I vantaggi della registrazione sono ovvi: essa è semplice, non richiede nessuna assistenza hardware e mantiene tutti gli ingressi e le uscite sincroni con il funzionamento del programma. Il suo svantaggio è altrettanto ovvio: la maggior parte del tempo del microprocessore è sciupato osservando dispositivi che non richiedono servizio. Inoltre il microprocessore può fornire il servizio ad un dispositivo troppo tardi, sciupando così molto tempo.

È perciò desiderabile un altro meccanismo che garantisca l'utilizzazione del tempo del processore per eseguire calcoli pratici piuttosto che la registrazione di dispositivi non richiesti tutte le volte. Comunque si sottolinea che la registrazione viene usata estensivamente quando un processore non ha nient'altro di meglio da fare e che essa mantiene semplice l'organizzazione globale. Si esaminerà ora l'alternativa principale alla registrazione: gli interrupt.

## Interrupt

Il concetto di interrupt è illustrato in figura 6-18. È disponibile una speciale linea hardware, la linea interrupt, che è connessa ad un pin specializzato del microprocessore.

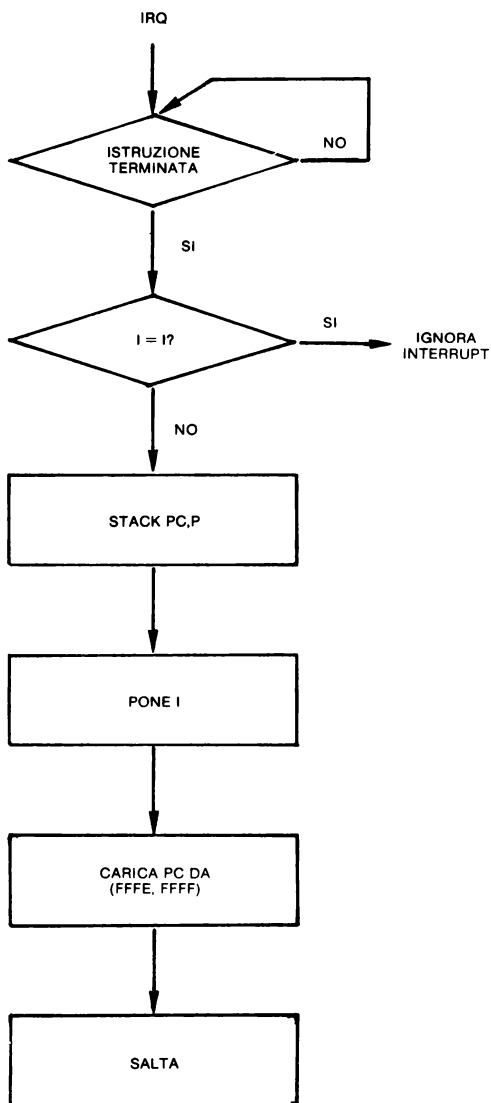



Figura 6.22: Elaborazione dell'interrupt

I dispositivi d'ingresso/uscita multipli possono essere connessi a questa linea di interrupt. Quando uno qualsiasi di questi richiede servizio esso invia un livello oppure un impulso su questa linea. Un segnale di interrupt è la richiesta di servizio da un dispositivo d'ingresso/uscita al processore. Si esaminerà ora la risposta del processore a questo interrupt.

In ogni caso il processore completa l'istruzione che stava eseguendo od anche che potrebbe creare confusione all'interno del microprocessore. Successivamente il microprocessore opera la diramazione ad una routine di manipolazione di interrupt che elaborerà l'interrupt stesso. La diramazione a tale subroutine implica che i contenuti del contatore di programma devono essere conservati nello stack. Un *interrupt deve perciò causare l'immagazzinamento automatico del contatore di programma nello stack.* Inoltre il registro di stato (P) dovrebbe essere automaticamene preservato poichè i suoi contenuti saranno alterati da qualsiasi istruzione successiva. Infine se la routine di manipolazione interrupt modificasse qualsiasi registro interno, questo dovrebbe essere automaticamente preservato nello stack.

Dopo che questi registri sono stati preservati si può operare la diramazione all'appropriato indirizzo di manipolazione interrupt. Alla fine di questa routine tutti i registri saranno ri-immagazzinati ed uno speciale ritorno da interrupt verrà eseguito cosicchè il programma principale riassuma l'esecuzione. Si esamineranno in maggior dettaglio le due linee di interrupt del 6502.

**Interrupt del 6502**

Il 6502 è equipaggiato con due linee di interrupt IRQ ed NMI. IRQ è la linea di interrupt regolare mentre NMI è un interrupt non mascherabile a priorità più elevata. Si esaminerà questo funzionamento.

IRQ è l'interrupt a livello attivato. Lo stato della linea IRQ sarà rivelato oppure ignorato dal microprocessore dipendentemente dal valore del suo flag interno I (flag della maschera interrupt). Si assumerà inizialmente che gli interrupt siano abilitati. Ogni volta che IRQ sarà attivato l'interrupt sarà rivelato dal microprocessore. Non appena l'interrupt è accettato (dopo il completamento dell'istruzione in corso di esecuzione), il flag interno I è posto ad "1" automaticamente.Questo preverrà un'ulteriore interruzione del microprocessore quando si sta manipolando i registri interni. Il 6502 quindi preserva automaticamente i contenuti di PC (il contatore di programma) e P (il registro di stato) nello stack.

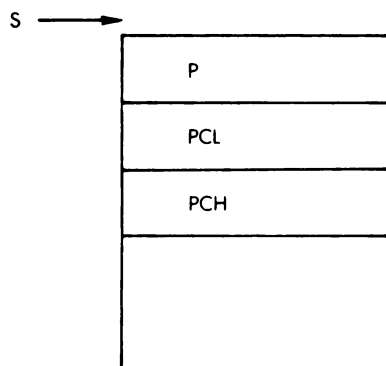



Figura 6.23: Stack del 6502 dopo interrupt

L'aspetto dello stack dopo che è elaborato un interrupt è illustrato in Figura 6-23.

Successivamente il 6502 preleverà automaticamente il contenuto delle locazioni di memoria "FFFE" ed "FFFF". Questa locazione di memoria a 16 bit conterrà il *vettore-interrupt.* Il 6502 preleverà i contenuti di questo indirizzo e quindi opererà la diramazione all'indirizzo specificato dal vettore a 16 bit. L'utente è responsabile della deposizione di questo indirizzo di vettore ad "FFFE" - "FFFF". Comunque diversi dispositivi

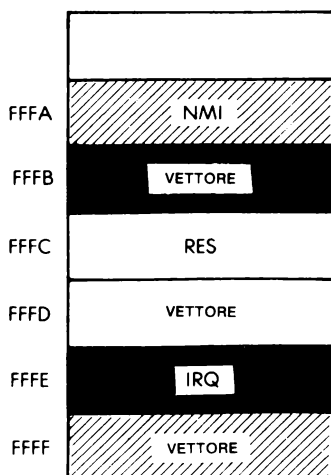



Figura 6.24: Vettori interrupt

possono essere connessi alla linea IRQ. In questo caso si avrà la diramazione ad una singola routine di manipolazione di interrupt. Come si può differenziare tra i vari dispositivi?

Questo sarà studiato al paragrafo successivo.

L'interrupt NMI è essenzialmente identico ad IRQ eccetto che esso non può essere mascherato dal bit I. È un interrupt a priorità più elevata usato tipicamente per i guasti di alimentazione. Il suo funzionamento è altrimenti identico eccetto che il processore opera automaticamente la diramazione ai contenuti di FFFA" - "FFFB". Questo è illustrato in Figura 6-24.

Il ritorno da interrupt è eseguito dall'istruzione RTI. Questa istruzione ritrasferisce nel microprocessore le tre parole di sommità dello stack che contengono P e PC (il contatore di programa a 16 bit). Il programma che era stato interrotto può quindi essere riassunto. Lo stato interno della macchina è esattamente identico a quello che si ha all'istante in cui si è verificato l'interrupt. L'effetto è stato quindi di introdurre un ritardo nell'esecuzione di un programma.

Prima del ritorno da interrupt il programmatore è responsabile di chiarire che l'interrupt è stato asservito e del ri-immagazzinamento del flag di disabilitazione interrupt. Inoltre se la routine di manipolazione interrupt modificasse i contenuti di qualsiasi registro come X od Y, il programmatore è specificamente responsabile per preservare questi registri nello stack prima dell'esecuzione della routine di manipolazione interrupt. Diversamente i contenuti di questi registri saranno modificati e quando il programma interrotto riassumerà l'esecuzione essi non saranno corretti.

Assumendo che la routine di manipolazione utilizzi i registri A, X ed Y, saranno necessarie cinque istruzioni all'interno del manipolatore di interrupt per preservare questi registri. Esse sono:

```
SAVAXY    PHA    INTRODUCE A NELLO STACK
          TXA    TRASFERISCE X AD A
          PHA    LO INTRODUCE
          TYA    TRASFERISCE Y AD A
          PHA    LO INTRODUCE
```

Sfortunatamente il 6502 può soltanto introdurre direttamente i contenuti di A o P nello stack. Ne risulta che preservare X ed Y vuol dire impiegare tempo: questo richiede 4 istruzioni.
Questo è illustrato in Figura 6-25.

Dopo il completamento della routine di manipolazione interrupt questi registri devono essere ri-immagazzinati ed il manipolatore di

interrupt termina con la sequenza di sei istruzioni.

```
PLA    ESTRAE Y DALLO STACK
TAY    RI-IMMAGAZZINA Y
PLA    ESTRAE X
TAX    RI-IMMAGAZZINA X
PLA    RI-IMMAGAZZINA A
RTI    USCITA
```

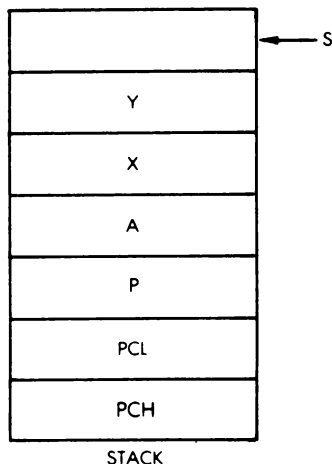



Figura 6.25: Conservazione di tutti i registri

**Esercizio 6-21:** *Utilizzando la tabella che indica il numero di cicli per istruzione, riportata in appendice, si calcoli quanto tempo si impiegherà per salvare e quindi ri-immagazzinare i registri A, X ed Y.*

Per un confronto grafico del processo di registrazione in funzione di quello di interrupt si faccia riferimento alla Figura 6-18 dove il processo di registrazione è illustrato in alto ed il processo di interrupt in basso. Si può vedere che, nella tecnica di registrazione, il programma impiega molto tempo in attesa. Utilizzando gli interrupt il programma viene interrotto, l'interrupt viene asservito e quindi ripristinato il programma. Comunque lo svantaggio ovvio di un interrupt è di introdurre alcune istruzioni addizionali all'inizio ed alla fine risolvendosi in un ritardo prima che possa esere eseguita la prima istruzione del dispositivo manipolatore. Questo è un altro svantaggio meno evidente.

Avendo chiarito il funzionamento delle due linee di interrupt si considerino ora due problemi importanti:

1. Come si risolve il problema di dispositivi multipli che fanno scattare un interrupt allo stesso istante?
2. Come si risolve il problema di un interrupt che si verifica mentre si sta asservendo ad un altro interrupt?

### Dispositivi Multipli Connessi ad una Singola Linea di Interrupt

Ogni volta che si verifica un interrupt il processore opera automaticamente la diramazione ad un indirizzo contenuto in "FFFE-FFFF" (per un IRQ) o ad "FFFA-FFFB" (per un NMI). Prima che esso possa fare qualsiasi elaborazione effettiva la routine di manipolazione interrupt deve determinare qual'è il dispositivo che fa scattare l'interrupt. Sono disponibili due metodi per identificare il dispositivo nei casi più comuni: un metodo software ed un metodo hardware.

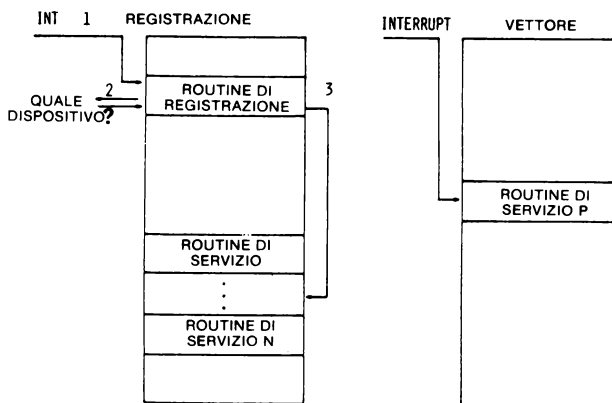



Figura 6.26: Interrupt registrato in funzione dell'interrupt mediante vettore

Nel metodo software viene utilizzato il metodo di registrazione: il microprocessore interroga a turno ciascun dispositivo e chiede: "Hai fatto scattare l'interrupt?". Se no, esso interroga quello successivo. Questo processo è illustrato in figura 6-26. Un programma campione è:

```
LDA    STATUS 1
BMI    ONE
LDA    STATUS 2
BMI    TWO
```

Il metodo hardware utilizza componenti addizionali ma fornisce immediatamente l'indirizzo del dispositivo che richiede l'interrupt, assume alla stessa richiesta di interrupt. Il dispositivo ora universalmente utilizzato per fornire questa possibilità è chiamato "PIC" ovvero controllore della priorità di interrupt. Tale PIC posizionerà automaticamente sul bus dati il richiesto indirizzo effettivo di diramazione per la periferica che richiede interrupt. Quando il 6502 andrà ad "FFFE-FFFF" preleverà questo vettore indirizzo. Questo concetto è illustrato in Figura 6-26.

Nella maggior parte dei casi la velocità di reazione ad un interrupt non è cruciale e viene utilizzato un approccio a registrazione. Se invece il tempo di risposta è una considerazione primaria occorre utilizzare un approccio hardware.

Figura 6.27: Diversi dispositivi possono utilizzare la stessa linea di interrupt

## Interrupt Simultanei

L'altro problema che può verificarsi è che un nuovo interrupt possa essere fatto scattare durante l'esecuzione di una routine di manipolazione interrupt. Si esaminerà cosa accade e come viene utilizzato lo stack per risolvere il problema. È stato indicato al Capitolo 2 che questo è un altro ruolo essenziale dello stack ed è giunto il momento di dimostrare il suo impiego. Ci si riferirà alla Figura 6-28 per illustrare gli interrupt multipli. Nell'illustrazione il tempo trascorre andando da sinistra a destra. I contenuti dello stack sono mostrati in fondo all'illustrazione. Guardando a sinistra, all'istante T0, è in esecuzione il programma P. Spostandosi a destra, all'istante T1, si verifica l'interrupt I1. Si assumerà che la maschera di interrupt sia abilitata, autorizzando così T1. Il programma P sarà sospeso. Questo è mostrato in fondo all'illustrazione. Lo stack conterrà il contatore di programma ed il registro di stato del programma P, almeno, più qualsiasi registro a scelta che deve essere conservato dal manipolatore di interrupt o da I1 stesso.

All'istante T1 inizia l'esecuzione dell'interrupt I1 fino all'istante T2. All'istante T2 si verifica l'interrupt I2. Si assumerà che l'interrupt I2 abbia una priorità più elevata dell'interrupt I1. Se esso avesse una

priorità più bassa sarebbe ignorato fino a che I1 non è stato completato. All'istante T2 i registri per I1 sono depositati nello stack e questo appare in fondo all'illustrazione.
Anche i conteggi del contatore di programma e di P sono introdotti nello stack. Inoltre la routine di I2 deve decidere se immagazzinare alcuni registri addizionali. I2 sarà eseguito fino al completamento fino all'istante T3.

Quando I2 termina i contenuti alla sommità dello stack sono estratti automaticamente nel 6502 e questo è illustrato in fondo alla Figura 6-28.

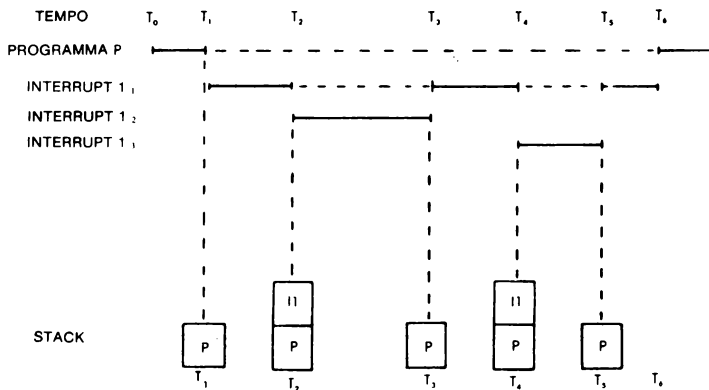



Figura 6.28: Stack durante gli interrupt

Così automaticamente, l'interrupt I1 riassume l'esecuzione. Sfortunatamente, all'istante T4 si verifica ancora un altro interrupt a priorità più alta. Si può vedere in fondo all'illustrazione che i registri per I1 vengono nuovamente introdotti nello stak. L'interrupt I3 è eseguito da T4 a T5 e termina a T5. In questo istante i contenuti dello stack sono estratti dal 6502 e l'interrupt I1 riassume l'esecuzione. Questa volta si ha il completamento ed esso termina a T6. A T6 i registri rimanenti che sono stati conservati nello stack sono estratti dal 6502 ed il programma P può riassumere l'esecuzione. Il lettore verificherà che lo stack è vuoto a questo punto. Infatti il numero di linee tratteggiate, indicanti la sospensione del programma, mostra allo stesso tempo quanti livelli si trovano nello stack.

**Esercizio 6-22:** *Se si assume che ogni volta che si verifica un interrupt siano conservati il contatore di programma PC, il registro P e l'accumulatore questi impiegheranno almeno quattro locazioni. (In pratica X ed Y necessitano spesso di essere conservati e si utilizzano sei locazioni). Assumendo*

*perciò che nello stack siano conservati soltanto tre registri quanti livelli di interrupt consente il 6502? (Si ricordi che lo stack è limitato a 256 locazioni all'interno della Pagina 1).*

**Esercizio 6-23:** *Assumendo questa volta che nello stack possono essere preservati 5 registri, qual'è il massimo numero di interrupt simultanei che possono essere manipolati? Quale altro fattore contribuirà a ridurre ulteriormente il numero di interrupt contemporanei?*

Si deve sottolineare comunque che in pratica i sistemi a microcalcolatore sono normalmente connessi ad un piccolo numero di dispositivi utilizzanti gli interrupt. Quindi è improbabile che in tale sistema si verifichi un numero elevato di interrupt contemporanei.

Sono stati ora risolti tutti i problemi normalmente associati con gli interrupt. Il loro impiego è, infatti, semplice e dovrebbero essere utilizzati vantaggiosamente anche dai nuovi programmatori. Si completerà l'analisi delle risorse del 6502 introducendo un ulteriore istruzione i cui effetti sono identici ad un interrupt sincrono.

### Break

Il comando BRK nel 6502 è l'equivalente di un interrupt software. Esso può essere inserito nel corso di un programma e si risolve, proprio come nel caso IRQ, nel salvataggio automatico di PC e P ed in una diramazione indiretta ad "FFFE-FFFF". Questa istruzione può essere vantaggiosamente utilizzata per generare interrupt programmati durante il collaudo di un programma. Questo originerà la creazione di punti di diramazione, arrestando il programma ad una determinata locazione ed operando la diramazione ad una routine che consentirà tipicamente all'utente di analizzare il programma. Poichè l'effetto netto di un break e di un interrupt è identico dopo la loro esecuzione occorre fornire un significato al programmatore per determinare se è stato utilizzato un interrupt oppure un break. Il 6502 porrà il flag B del registro P (conservato nello stack) ad "1" se era un break ed a "0" se era un interrupt. La verifica dello stato di questo bit può essere eseguita dal semplice programma seguente:

```
BTEST   PLA           LEGGE IN A LA SOMMITA' DELLO STACK
        PHA           RISCRIVE
        AND# $10      MASCHERA DEL BIT B
        BNE BRKPRG    VA AL PROGRAMMA BREAK
```

Questo programma di verifica viene normalmente inserito alla fine della sequenza di registrazione che determina la natura del dispositivo che fa scattare l'interrupt.

Attenzione: una caratteristica del break è di preservare automaticamente i contenuti del contatore di programma *più 2.* Poichè il break è una istruzione di un solo byte il programmatore deve talvolta aggiustare i contenuti del contatore di programma nello stack utilizzando un'istruzione di incremento o decremento per riassumere l'esecuzione dell'indirizzo corretto. In particolare il break è utilizzato estensivamente durante il collaudo scrivendolo su un'altra istruzione del programma. Se il programma è riassemblato prima dell'esecuzione, i contenuti del contatore di programma che sono stati salvati dovranno essere normalmente incrementati di 1.

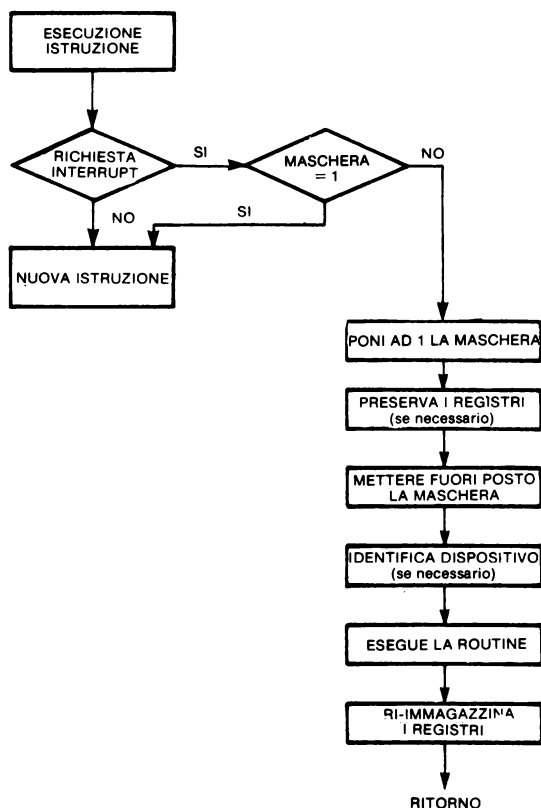



Figura 6.29: Logica dell'interrupt

## SOMMARIO

È stato presentato in questo capitolo l'insieme delle tecniche utilizzate per comunicare col mondo esterno. Dalle routine elementari d'ingresso/uscita ai programmi più complessi per comunicare con periferiche effettive, si è imparato a sviluppare tutti i programmi usuali e si è anche esaminata l'efficienza di programmi tipici nel caso di un trasferimento parallelo e di una conversione da parallelo a seriale. Infine si è imparato a classificare il funzionamento di periferiche multiple utilizzando registrazione e d'interrupt. Naturalmente molti altri dispositivi d'ingresso/uscita devono essere connessi al sistema. Con l'insieme delle tecniche che sono state presentate e con la comprensione delle periferiche coinvolte è possibile risolvere la maggior parte dei problemi comuni.

Al capitolo successivo si esamineranno le caratteristiche effettive dei chip di interfaccia ingresso/uscita normalmente connessi al 6502. Quindi si considereranno le strutture dati di base di cui il programmatore deve conoscere l'utilizzazione.

## ESERCIZI

**Esercizio 6-24:** *Un display LED a 7 segmenti può mostrare anche digit diversi da quelli dell'alfabeto esadecimale. Si calcolino i codici di: H, I, J, L, O, P, S, U, Y, g, h, i, j, l, n, o, p, r, t, u, y.*

**Esercizio 6-25:** *Il diagramma di flusso per la direzione di interrupt appare nella Figura 6-29. Si risponda alle seguenti domande:*
*a- Che cosa è fatto dall'hardware e cosa dal software?*
*b- Qual'è l'impiego della maschera?*
*c- Quanti registri dovrebbero essere preservati?*
*d- Come viene identificato il dispositivo che origina l'interrupt?*
*e- Cosa fa l'istruzione RTI? In cosa differisce da un ritorno da subroutine?*
*f- Si suggerisca un modo per manipolare una situazione di overflow dello stack.*
*g- Qual'è lo svantaggio ("tempo perso") introdotto dal meccanismo di interrupt?*

CAPITOLO 7

# DISPOSITIVI D'INGRESSO/USCITA

## INTRODUZIONE

Si è imparato come programmare il microprocessore 6502 nella maggior parte delle situazioni più comuni. Comunque è necessaria una trattazione particolare dei chip d'ingresso/uscita normalmente connessi al microprocessore. A causa del progresso dell'integrazione LSI sono stati introdotti nuovi chip prima inesistenti. Ne risulta che la programmazione di un sistema richiede, naturalmente, prima di programmare, il microprocessore stesso e poi anche di *programmare i chip d'ingresso/uscita*. Infatti è spesso molto più difficoltoso ricordare come programmare le varie scelte di controllo di un chip d'ingresso/uscita che programmare il microprocessore stesso! Questo non perchè la programmazione in sè è più difficoltosa ma perchè ciascuno di questi dispositivi ha le sue eccentricità.
Si esaminerà di seguito prima il dispositivo d'ingresso/uscita più generale, il chip d'ingresso/uscita programmabile (in breve un "PIO") e poi i "miglioramenti" di questo PIO convenzionale ora utilizzato frequentemente con il 6502: il 6520, 6530, 6522 e 6532.

## Il PIO CONVENZIONALE (6520)

Non esiste il "PIO Convenzionale". Comunque il dispositivo 6520 è essenzialmente analogo nella funzione a tutti i PIO similari prodotti dagli altri costruttori per lo stesso scopo. Lo scopo di un PIO è di fornire una connessione multiporta per i dispositivi d'ingresso/uscita. (Una "porta" è semplicemente un set di 8 linee d'ingresso/uscita). Ogni PIO fornisce almeno due set di linee ad 8 bit per dispositivo I/O. Ciascun dispositivo I/O richiedè un *buffer* dati per stabilizzare i contenuti del bus dati almeno in uscita. Perciò il PIO sarà equipaggiato almeno di un buffer per ogni porta.

Inoltre è stato stabilito che il microcalcolatore utilizzerà una procedura *handshaking* od anche *interrupt* per comunicare con il dispositivo I/O. Il PIO utilizzerà anche una procedura simile per comunicare con la

periferica. Ogni PIO deve, perciò, essere equipaggiato con almeno *due linee di controllo* per porta per realizzare la funzione handshaking.

Il microprocessore richiederà anche la capacità di leggere lo stato di ciascuna porta. Ogni porta deve essere equipaggiata con uno o più bit di stato. Infine all'interno di ogni PIO esisterà un certo numero di scelte per configurare le sue risorse. Il programmatore deve essere in grado di accedere al registro speciale all'interno del PIO per specificare le scelte di programmazione. Questo è il *registro di controllo.* Nel caso del 6502 l'informazione di stato è parte del registro di controllo.

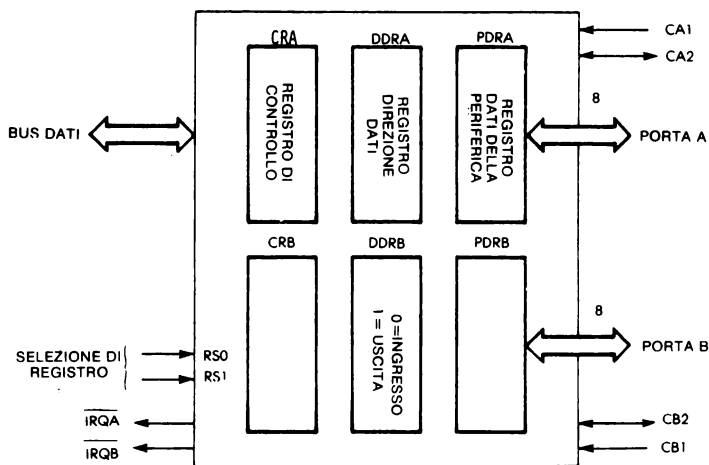



Figura 7.1: PIO tipico

Una caratteristica essenziale del PIO è il fatto che ogni linea deve essere configurata come linea d'ingresso oppure d'uscita. Lo schema di un PIO appare in Figura 7-1. Il programmatore deve specificare per qualsiasi linea se sarà d'ingresso o d'uscita. Per programmare la direzione di una linea viene fornito un *registro di direzione dati* per ciascuna porta. Uno "0" in una posizione di bit del registro di direzione dati specifica un ingresso. Un "1" specifica un'uscita.

Può essere sorprendente osservare che uno "0" è utilizzato per un ingresso ed un "1" per un'uscita quando in realtà "0" dovrebbe corrispondere all'Uscita ed "1" all'Ingresso. Questo è abbastanza deliberato: ogni volta che viene applicata l'alimentazione al sistema è di grande importanza che tutte le linee I/O siano configurate come *ingresso.*

Diversamente se il microcalcolatore è connesso ad alcune periferiche guaste esso potrebbe attivarle accidentalmente. Quando viene applicato un reset tutti i registri sono normalmente azzerati e questo risulterà nella configurazione come ingressi di tutte le linee del PIO. La connessione al microprocessore appare sulla sinistra dell'illustrazione. Il PIO connette naturalmente al bus dati ad 8 bit, al bus indirizzi del microprocessore ed al bus di controllo del microprocessore. Il programmatore specificherà semplicemente l'indirizzo di qualsiasi registro cui si desidera accedere all'interno del PIO. Il 6522, che è compatibile col Motorola 6820, ha

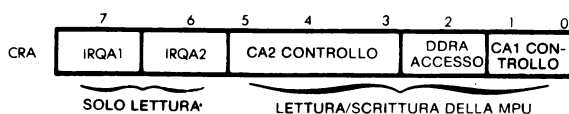



Figura 7.2: Formato della parola di controllo PIA

| RS1 | RS0 | CRA 2 | CRB 2 | REGISTRO SELEZIONATO |
|---|---|---|---|---|
| 0 | 0 | 1 | - | REGISTRO DELLA PERIFERICA A |
| 0 | 0 | 0 | - | REGISTRO DIREZIONE DATI A |
| 0 | 1 | - | - | REGISTRO DI CONTROLLO A |
| 1 | 0 | - | 1 | REGISTRO DELLA PERIFERICA B |
| 1 | 0 | - | 0 | REGISTRO DIREZIONE DATI B |
| 1 | 1 | - | - | REGISTRO DI CONTROLLO B |

Figura 7.3: Registri PIA di indirizzamento

ereditato una peculiarità: esso è equipaggiato con 6 registri interni. Comunque si può specificare solo un registro su *quattro!* Questo problema viene risolto commutando la posizione di bit 2 del registro di controllo. Quando questo bit è uno "0" si può selezionare il corrispondente registro di direzione dati. Quando è un "1" può essere selezionato il registro dati. Perciò ogni volta che il programmatore desidera scrivere dati nel registro di direzione dati dovrà prima assicurarsi che il bit 2 dell'appropriato registro di controllo sia zero, prima di poter selezionare questo registro. Questo è talvolta inadatto per il programma ma è importante da ricordare per evitare grosse difficoltà.

Per chiarire l'effetto della selezione di indirizzo sul 6520 viene riportata la tabella di selezione indirizzo.

RS0 ed RS1 sono due segnali di selezione registro che sono stati derivati dal bus dati. In altre parole essi rappresentano due bit dell'indirizzo specificato dal programmatore. CRA è il registro di controllo della porta A. CRA (2) è il bit 2 di questo registro. CRB è il registro di controllo della porta B.

### Il Registro di Controllo Interno

Il registro di controllo del 6520 specifica, come si è visto nella posizione di bit 2, un modo di selezione per i registri interni della porta. Inoltre esso fornisce un certo numero di opzioni per la generazione o rivelazione di interrupt e per la realizzazione di funzioni handshake automatiche.
La descrizione completa delle caratteristiche disponibili non è necessaria in questa sede. L'utente di qualsiasi sistema pratico utilizzante il 6520 farà semplicemente riferimento al data-sheet che mostra l'effetto del posizionamento dei vari bit del registro di controllo. Ogni volta che il sistema è inizializzato il programma dovrà caricare il registro di controllo del 6520 con i contenuti corretti per la specifica applicazione.

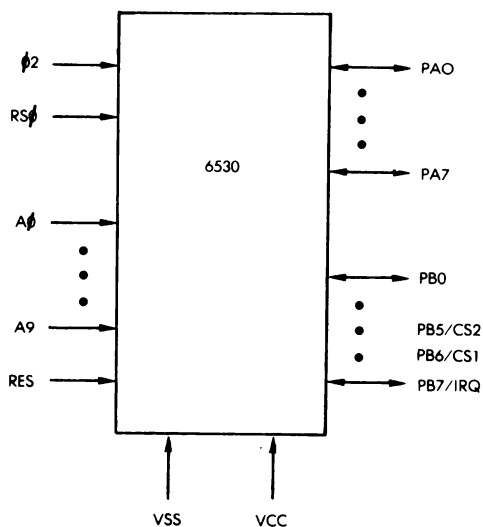



Figura 7.4: Pin d'uscita del 6530

## Il 6530

Il 6530 realizza una combinazione di quattro funzioni: RAM, ROM, PIO e TIMER. La RAM è una memoria 64 x 8. La ROM è una memoria di 1 k x 8, il timer fornisce al programmatore le possibilità di temporizzazione ad intervallo multiplo. La parte PIO è essenzialmente analoga al 6520, precedentemente descritto: ci sono due porte; ogni porta ha un registro dati ed un registro di direzione dati più un registro di comando. Uno "0" in una data posizione di bit del registro di direzione specifica un ingresso, mentre un "1" specifica un'uscita.

Il timer ad intervallo programmabile può essere programmato per contare fino a 256 intervalli (esso internamente ha 8 bit). Il programmatore può specificare che il periodo di tempo sia 1, 8, 64 oppure 1024 volte il clock del sistema. Ogni volta che viene raggiunto il conteggio il flag interrupt del chip sarà posto al valore logico "1". I contenuti del timer sono posizionati per mezzo del bus dati. I quattro possibili intervalli di tempo devono essere specificati sulle linee A0 ed A1 del bus indirizzi.

Tre pin della porta B hanno un ruolo duale: PB5, PB6 e PB7 possono essere utilizzati per funzioni di controllo. Il pin PB7, per esempio, può essere programmato come un ingresso interrupt.

Questo chip è particolarmente utilizzato nella scheda KIM (si noti che sulla KIM, PB6 non è disponibile).

### Programmazione di un PIO

Come esempio si riporta un programma che impiega un 6520 oppure un 6522 (si assume che il registro di controllo sia già stato posizionato).

```
LDA    # FF    PONE DIREZIONE DATI
STA    DDRB    CONFIGURA B COME USCITA
LDA    # 00
STA    IORB    GENERA L'USCITA ZERO
```

DDRB è l'indirizzo del Registro di Direzione Dati della porta B di questo PIO. IORB è l'Ingresso-Uscita o Registro Dati della porta B, "FF" esadecimale è "11111111" binario = tutte uscite.

## Il 6522

Il 6522, detto anche "adattatore di interfaccia versatile" (VIA), è una versione migliorata del 6520. Oltre alle possibilità del 6520, esso fornisce due timer ad intervallo programmabile ed un convertitore serie-parallelo più parallelo-serie oltre al latch dei dati d'ingresso. La descrizione hardware dettagliata di questo componente è oltre lo scopo di questo libro.

Figura 7.5: Impiego del PIA: caricamento registro di controllo

Figura 7.6: Impiego del PIA: caricamento direzione dati

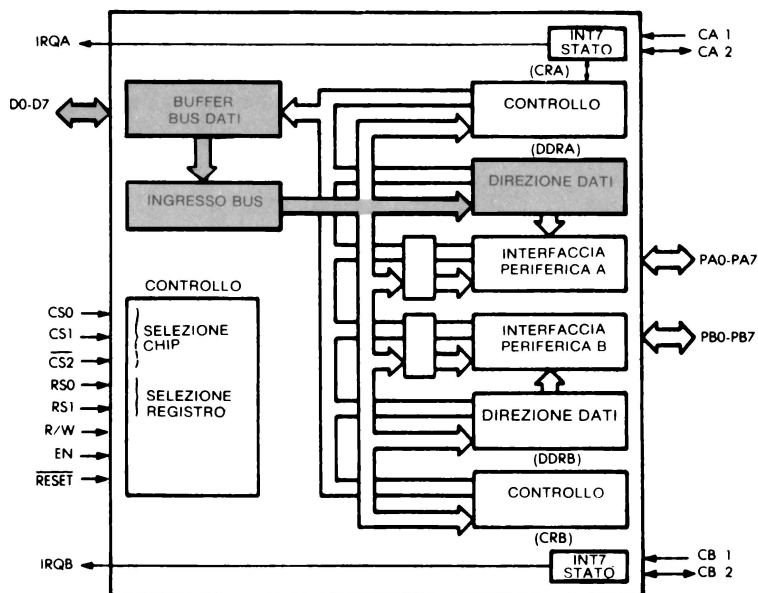
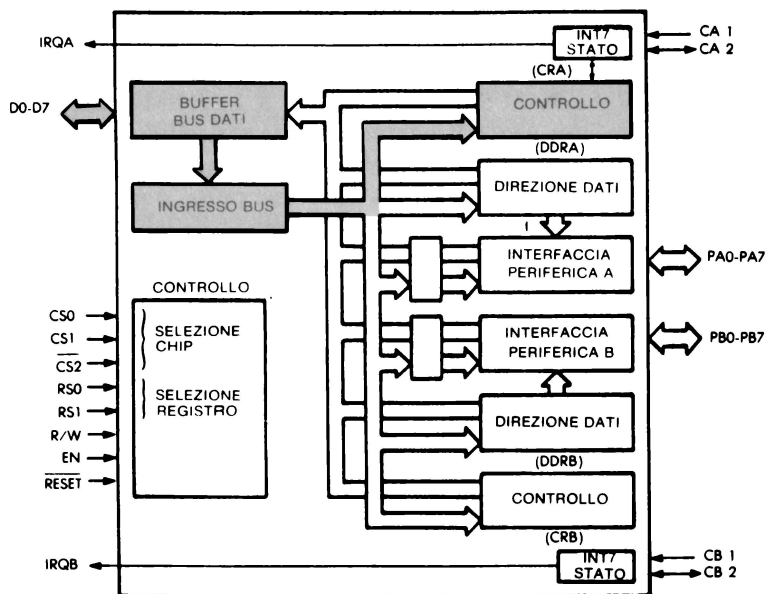
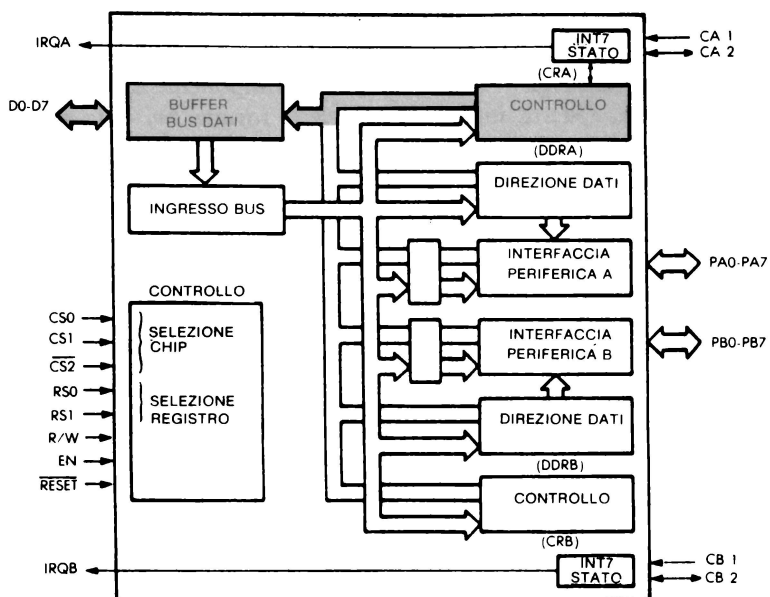



Figura 7.7: Impiego del PIA: lettura stato

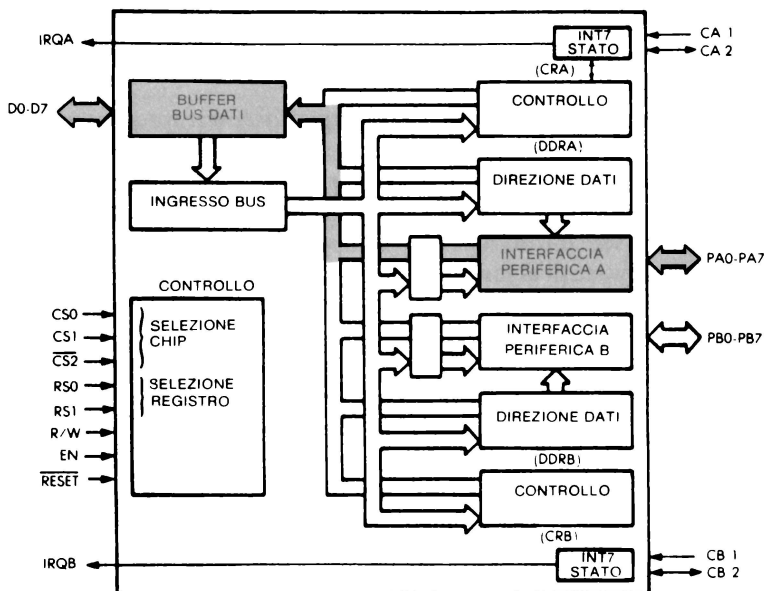



Figura 7.8: Impiego del PIA: lettura ingresso

Semplicemente con la descrizione fornita per i precedenti componenti dovrebbe essere semplice per il programmatore familiarizzarsi con l'indirizzamento dei registri interni di questo componente e della sua programmazione. Questa informazione viene fornita nei data-sheet del costruttore.

### Il 6532

Il 6532 è un chip combinazione che comprende una RAM 128 x 8, un PIO con due porte bidirezionali ed un timer ad intervallo programmabile. Esso viene utilizzato nella scheda SYM, costruita dalla Synertek Systems, analoga alla scheda KIM costruita dalla MOS Technology e dalla Rockwell. Anche qui l'utente dovrebbe esaminare attentamente i data-sheet di questo componente per imparare come indirizzare ed utilizzare i vari registri interni.

## SOMMARIO

Sfortunatamente, per rendere effettivo l'impiego di tali componenti, sarà necessrio capire in dettaglio la funzione di ogni bit, o gruppo di bit, all'interno dei vari registri di controllo. Questi nuovi chip complessi rendono automatiche molte procedure che precedentemente venivano eseguite mediante software oppure mediante logica speciale. In particolare molte delle procedure di handshaking sono automatizzate all'interno dei componenti come il 6522.
Inoltre alcune manipolazioni e rivelazioni di interrupt possono essere interne. Con le informazioni presentate al capitolo precedente il lettore dovrebbe essere in grado di esaminare i data sheet corrispondenti e capire quali sono le funzioni dei vari segnali e registri. Naturalmente stanno per essere introdotti nuovi componenti che offriranno una realizzazione hardware di algoritmi ancora più complessi. Anche qui il lettore dovrebbe essere in grado di capire studiando attentamente i data-sheet del costruttore.

CAPITOLO 8

# ESEMPI DI APPLICAZIONE

## INTRODUZIONE

Questo capitolo ha lo scopo di verificare l'abilità alla programmazione poichè presenta una certa quantità di programmi di utilità pratica. Questi programmi o "routine" si incontrano frequentemente nelle applicazioni e sono generalmente chiamati "utility routines".
Essi richiederanno una sintesi delle conoscenze e delle tecniche presentate fino ad ora.
Si procederà al prelievo di caratteri da un dispositivo I/O e ad elaborarli in vario modo. Prima però è necessario azzerare un'area della memoria (questo potrebbe non essere necessario ma ciascuno di questi programmi è presentato soltanto come esempio di programmazione).

## AZZERAMENTO DI UNA PARTE DELLA MEMORIA

Si vuole azzerare i contenuti della memoria dall'indirizzo BASE + 1 all'indirizzo BASE + LENGHT, dove LENGHT è minore di 256.

Il programma è:

```
ZEROM    LDX # LENGTH
         LDA # 0
CLEAR    STA BASE, X
         DEX
         BNE CLEAR
         RTS
```

Si noti che il registro X è utilizzato come indice per puntare alla locazione corrente della parte di memoria da azzerare.
L'accumulatore A è caricato solo una volta con il valore 0 (tutti zeri) e quindi trascritto alle locazioni di memoria successive:
BASE + LENGTH, BASE + LENGTH — 1, ecc. finchè X è decrementato a 0. Quando X = 0 si ha il ritorno dal programma.

Per esempio, in una verifica delle funzioni di una memoria, questo programma potrebbe essere utilizzato per azzerare un blocco e quindi per verificare i suoi contenuti.

**Esercizio 8-1:** *Si scriva un programma di verifica di memoria che azzeri un blocco di 256 parole e quindi verifichi che ogni locazione è 0. Quindi esso scriverà tutti uni e verificherà il contenuto del blocco. Quindi esso scriverà 01010101 e verificherà i contenuti.*

Si registreranno ora i dispositivi I/O per vedere se uno di essi richiede servizio.

## POLLING DEI DISPOSITIVI DI I/O

Si assumerà che al sistema in esame siano connessi 3 dispositivi I/O. I loro registri di stato siano localizzati agli indirizzi IOSTATUS1, IOSTATUS2, IOSTATUS3.
Se il loro bit di stato è nella posizione di bit 7 si leggerà il registro di stato e si verificherà il bit segno. Se il bit di stato è dovunque si trarrà vantaggio dall'istruzione BIT del 6502:

```
TEST        LDA         MASK
            BIT         IOSTATUS1
            BNE         FOUND1
            BIT         IOSTATUS2
            BNE         FOUND2
            BIT         IOSTATUS3
            BNE         FOUND3
            (uscita in caso di errore)
```

Se si verifica la posizione di bit 7 la MASK conterrà per esempio "00100000". Come risultato dell'istruzione BIT il bit 2 del flag di stato sarà posto ad 1 se "MASK AND IOSTATUS" non è zero, cioè se il bit corrispondente ad IOSTATUS è uguale a quello corrispondente di MASK. L'istruzione BNE (opera diramazione se non uguale zero) quindi si risolverà in una diramazione all'appropriata routine FOUND.

## ACCETTAZIONE DEI CARATTERI ALL'INGRESSO

Si assuma di aver trovato che un carattere è disponibile sulla tastiera. Si accumulino i caratteri in un'area di memoria chiamata buffer finchè

non si incontra un carattere speciale chiamato SPC, il cui codice è stato precedentemente definito.

La subroutine GETCHAR preleverà un carattere dalla tastiera (vedere il capitolo 6 per ulteriori dettagli) e lo posizionerà nell'accumulatore. Si assume che al massimo saranno prelevati 256 caratteri prima di trovare un carattere SPC.

```
STRING  LDX   # 0         INIZIALIZZA L'INDICE A ZERO
NEXT    JSR   GETCHAR
        CMP   # SPC       È IL CARATTERE BRK?
        BEQ   OUT         SE SI TERMINA
        STA   BUFFER, X   NO: CONSERVA CARATTERE
        INX               INCREMENTA IL PUNTATORE
        JMP   NEXT        ACCETTA IL CARATTERE
                          SUCCESSIVO
OUT     RTS
```

**Esercizio 8-2:** *Si migliori questa routine di base:*

*a- Operi l'eco di un carattere di ritorno al dispositivo (per una telescrivente per esempio)*

*b- Verifichi che la stringa d'ingresso non sia più lunga di 256 caratteri*

Ora si ha una stringa di caratteri in un buffer della memoria. Si processeranno in vari modi.

## VERIFICA DI UN CARATTERE

Si determini se il carattere posizionato alla locazione di memoria LOC è uguale a 0,1 oppure 2:

```
ZOT     LDA   LOC
        CMP   # $00
        BEQ   ZERO
        CMP   # $01
        BEQ   ONE
        CMP   # $02
        BEQ   TWO
        JMP   NOT FND
```

Si legge semplicemente il carattere quindi si impiega l'istruzione CMP per controllare il suo valore.

Si esamina ora una verifica diversa.

## VERIFICA DI PARENTESI

Si determini se il carattere ASCII posizionato alla locazione di memoria LOC è una cifra tra 0 e 9:

```
BRACK     LDA    # $40
          ADC    # $40      FORZA L'OVERFLOW
          LDA    LOC
          ORA    # $80      PONE BIT 7 = 1
          CMP    # $B0      0 ASCII
          BCC    TOOLOW
          CMP    # $B9      9 ASCII
          BEQ    OUT        9 ESATTAMENTE
          BCS    TOOHIGH
OUT       CLC
          CLV
          RTS

TOOLOW    SEC               PONE C AD UNO
          CLV
          RTS
TOOHIGH   RTS               (C'È UNO)
```

0 ASCII è rappresentato in esadecimale da "B0"
9 ASCII è rappresentato in esadecimale da "B9"

Si ricordi che utilizzando un'istruzione CMP il bit carry sarà posto ad 1 se il valore letterale che segue è minore o uguale al valore dell'accumulatore. Esso sarà posizionato a 0 se maggiore.

Se B0 è maggiore del carattere, il carattere è troppo basso e si verifica una diramazione.

Quindi si confronta con il registro B9. Se esso è minore od uguale a 9 si esce. Diversamente si va TOOHIGH.

Quando si esce da questo programma si vuole conoscere se il numero è TOOLOW, TOOHIGH oppure tra 0 e 9.

Questo sarà indicato dai flag C e V. V non viene alterato da CMP. Invece CMP cambia Z, N e C.

Quando si ritorna da questa subroutine uno "0" in V indica "troppo alto", un "1" in C indica "troppo basso" ed uno "0" in C indica una cifra corretta tra 0 e 9.

Naturalmente le altre conversioni, come il caricamento di un digit nell'accumulatore, potrebbero essere utilizzate per indicare il risultato delle verifiche.

**Esercizio 8-3:** *Si semplifichi il programma precedente verificando rispetto al carattere ASCII che segue "9" invece di 9 esatto.*

**Esercizio 8-4:** *Si determini se un carattere ASCII contenuto nell'accumulatore è una lettera dell'alfabeto.*

Quando si utilizza una tabella ASCII si noterà che viene quasi sempre impiegata la parità. Per esempio l'ASCII di "0" è "0110000" cioè un codice a 7 bit. Comunque se si usa la parità dispari, per esempio, (si garantisce che il numero totale di uni in una parola sia dispari) allora il codice diviene "10110000". Un ulteriore "1" viene aggiunto a sinistra. Questo è "B0" in esadecimale. Si svilupperà ora un programma per generare la parità.

## GENERAZIONE DI PARITA'

Questo programma genererà una parità pari nella posizione di bit 7:

```
PARITY   LDX   # $07     CONTEGGIO DI BIT
         LDA   # $00
         STA   ONECNT    CONTEGGIO DI UNI
         LDA   CHAR      LETTURA DEL CARATTERE
         ROL   A         SCARICA IL BIT 7
NEXT     ROL   A         BIT SUCCESSIVO
         BCC   ZERO      È UN 1?
ONE      INC   ONECNT
ZERO     DEX             DECREMENTA IL CONTEGGIO
                         DI BIT
         BNE   NEXT      ULTIMO BIT?
         ROL   A         RI-IMMAGAZZINA IL BIT 0
         ROL   A         SCARICA IL BIT?
         LSR   ONECNT    IL BIT PIU' A DESTRA È LA PARITA'
         LSR   A         LO METTE IN A
         RTS
```

Il registro X è utilizzato per contare i bit mentre essi sono spostati a sinistra dell'accumulatore. Ogni volta che un "1" viene portato via dalla sinistra dell'accumulatore (mediante la verifica di BCC) il contatore di uni viene incrementato. Quando sono stati spostati 8 bit (il programma ignora il bit 7 che sarà il bit di parità) A viene fatto scorrere a sinistra

altre due volte cosicchè il bit 6 è a sinistra di A.
Il bit di parità corretto è il bit più a destra di ONECNT: esso viene posizionato nel bit carry da LSR e diviene il bit 7 di A. Un'altra istruzione LSR A ricopia questo bit nella posizione di bit 7 di A ed il problema proposto è risolto.

**Esercizio 8-5:** *Utilizzando il programma precedente come esempio si verifichi la parità di una parola. Si deve calcolare la parità corretta e quindi confrontarla con quella prevista dal programma.*

## CONVERSIONE DI CODICE: da ASCII a BCD

La conversione da ASCII a BCD è molto semplice. Si osserverà che la rappresentazione esadecimale dei caratteri ASCII da 0 a 9 va da B0 a B9. La rappresentazione BCD si ottiene perciò semplicemente eliminando la "B" cioè mascherando il nibble di sinistra (4 bit):

```
LDA     CHAR
AND     # SOF       MASCHERA IL NIBBLE DI SINISTRA
STA     BCDCHAR
```

**Esercizio 8-6:** *Si scriva un programma per convertire il BCD in ASCII.*

**Esercizio 8-7:** *(Più difficoltoso) Si scriva un programma per convertire il BCD in binario.*

Suggerimento: $N_3 N_2 N_1 N_0$ in BCD è $(((N_3 \times 10) + N_2) \times 10 + N_1) \times 10 + N_0$ in binario.
Per moltiplicare per 10 si impieghi lo spostamento a sinistra (=x2), un altro scorrimento a sinistra (=x4), un ADC (=x5) ed un altro scorrimento a sinsitra (=x10).
Nella notazione BCD intera la prima parola può contenere il conteggio dei digit BCD, il nibble successivo contiene il segno ed ogni nibble successivo contiene un digit BCD (non si considera il punto decimale). L'ultimo nibble del blocco può essere inutilizzato.

## RICERCA DELL'ELEMENTO MAGGIORE DI UNA TABELLA

L'indirizzo di partenza della tabella sia contenuto all'indirizzo di memoria BASE in pagina zero. Il primo ingresso della tabella è il numero di byte che essa contiene. Questo programma ricercherà l'elemento maggiore della tabella. Il suo valore sarà depositato in A e la sua posizione sarà immagazzinata alla locazione di memoria INDEX.

Questo programma utilizza i registri A ed Y ed impiegherà l'indirizzamento indiretto, cosicchè esso può ricercare qualsiasi tabella posizionata genericamente nella memoria.

```
MAX        LDY   # 0          QUESTO È L'INDICE ALLA TABEL-
                              LA
           LDA   (BASE), Y    INGRESSO ACCESSO 0 = LUNGHEZ-
                              ZA
           TAY                LO CONSERVA IN Y
           LDA   # 0          VALORE MASSIMO
                              INIZIALIZZATO A ZERO
           STA   INDEX        INIZIALIZZA L'INDICE A ZERO
LOOP       CMP   (BASE), Y    L'ELEMENTO ATTUALE È
                              IL MASSIMO?
           BCS   NOSWITCH     SI?
           LDA   (BASE), Y    CARICA IL NUOVO MASSIMO
           STY   INDEX        LOCAZIONE DEL MASSIMO
NOSWITCH   DEY                PUNTA AL NUOVO ELEMENTO
           BNE   LOOP         CONTINUA LA VERIFICA?
           RTS                FINITO SE Y = 0
```

Questo programma verifica prima l'n-esimo ingresso. Se questo è maggiore di 0 esso va in A e la sua locazione è memorizzata in INDEX. Quindi viene verificato l' (n-1)-esimo elemento, ecc. Questo programma lavora con interi positivi.

**Esercizio 8-8:** *Si modifichi il programma cosicchè esso lavori anche per numeri negativi in complemento a 2.*

**Esercizio 8-9:** *Questo programma lavorerà anche con caratteri ASCII?*

**Esercizio 8-10:** *Si scriva un programma che selezioni n numeri in ordine decrescente.*

**Esercizio 8-11:** *Si scriva un programma che scelga n nomi (di 3 caratteri ciascuno) in ordine alfabetico.*

## SOMMA DI N ELEMENTI

Questo programma calcolerà la somma a 16 bit degli n ingressi di una tabella. L'indirizzo di partenza della tabella è contenuto all'indirizzo di memoria BASE in pagina zero. Il primo ingresso della tabella contiene il numero N di elementi. La somma a 16 bit sarà depositata alle locazioni

di memoria SUMLO e SUMHI. Se la somma dovesse richiedere più di 16 bit, sarebbero conservati solo i 16 bit più bassi (si dice che i bit di ordine elevato sono stati troncati).
Questo programma modificherà i registri A ed Y. Esso considera al massimo 256 elementi.

```
            LDA  # 0          INIZIALIZZA SUM
            STA  SUMLO        INIZIALIZZA SUM
            STA  SUMHI        INIZIALIZZA SUM
            TAY               INIZIALIZZA Y A ZERO
            LDA  (BASE), Y    PONE N
            TAY               IN Y
            CLC               AZZERA CARRY PER ADC
ADLOOP      LDA  (BASE), Y    ACCETTA L'ELEMENTO
                              SUCCESSIVO
            ADC  SUMLO        LO SOMMA A SUMLO
            STA  SUMLO        CONSERVA IL RISULTATO
            BCC  NOCARRY      RIPORTO?
            INC  SUMHI        LO SOMMA A SUMHI
            CLC               ELEMENTO SUCCESSIVO PER
NOCARRY     DEY               LA SOMMA SUCCESSIVA
            BNE  ADLOOP       ANCORA SE Y NON È ZERO
            RTS
```

Questo programma è diretto ed autoesplicativo.

**Esercizio 8-12:** *Si modifichi questo programma per calcolare:*
*a- una somma a 24 bit*
*b- una somma a 32 bit*
*c- per rivelare qualsiasi overflow.*

## UN CALCOLO CHECKSUM

Un checksum è un digit od un insieme di digit calcolati da un blocco di caratteri successivi. La checksum viene calcolata all'istante in cui i dati sono immagazzinati e posizionata alla fine. Per verificare l'integrità dei dati la checksum viene ricalcolata e confrontata col valore immagazzinato. Una diversità indica un errore oppure un guasto.

Vengono utilizzati diversi algoritmi. In questo caso si opererà l'OR esclusivo di tutti i byte di una tabella di N elementi ed il risultato sarà depositato nell'accumulatore. come al solito la base della tabella è immagazzinata all'indirizzo BASE in pagina zero. Il primo ingresso della tabella è il numero di elementi N. Il programma modifica A ed Y. N deve essere minore di 256.

```
CHECKSUM    LDY    # 0        PUNTA AL PRIMO INGRESSO
            LDA    (BASE), Y  ACCETTA N
            TAY               LO IMMAGAZZINA IN Y
            LDA    # 0        INIZIALIZZA CHECKSUM
CHLOOP      EOR    (ADDR), Y  EOR INGRESSO SUCCESSIVO
            DEY               PUNTA AL SUCCESSIVO
            BNE    CHLOOP     PROSEGUE
            RTS
```

## CONTEGGIO DI ZERI

Questo programma conterà il numero di zeri di una tabella e lo depositerà nel registro X.
Esso modifica A, X, Y:

```
ZEROES      LDY    # 0        PUNTA AL PRIMO INGRESSO
            LDA    (ADDR), Y  ACCETTA N
            TAY               LO IMMAGAZZINA IN Y
            LDX    # 0        INIZIALIZZA IL NUMERO DI
                              ZERI
ZLOOP       LDA    (ADDR), Y  ACCETTA L'INGRESSO
                              SUCCESSIVO
            BNE    NOTZ       QUESTO È ZERO?
            INX               SI, LO CONTA
NOTZ        DEY               PUNTA AL SUCCESSIVO
            BNE    ZLOOP      PROSEGUE
            RTS
```

**Esercizio 8-13:** *Si modifichi questo programma per contare:*
*a- il numero di start (il carattere "*")*
*b- il numero di lettere dell'alfabeto*
*c- il numero di cifre tra 0 e 9*

## RICERCA DI UNA STRINGA

Si supponga che una stringa di caratteri, come indicato in Fig. 8-1, sia memorizzata in memoria. All'accorrenza si ricercherà la stringa per ricavarne una più breve detta template (TMPLT), di lunghezza TPTLEN. La lunghezza della stringa originale è STRLEN ed alla fine del programma il registro X conterrà la locazione in cui è stata trovata TEMPLT oppure FF esadecimale. La Fig. 8-2 mostra il diagramma di flusso per il programma. La stringa viene prima esplorata alla ricerca del primo carattere di TEMPLT. Se non viene trovato il primo carattere, si

uscirà dal programma. Se invece viene trovato il primo carattere, si confronta il secondo carattere con quello successivo della stringa. Se non sono uguali si riparte alla ricerca del primo carattere, poichè il primo carattere potrebbe essere ripetuto nella stringa originale. Se il primo ed il secondo sono uguali la ricerca procede con i caratteri successivi di TEMPLT in modo esattamente analogo. La Fig. 8-3 mostra il programma corrispondente. Si noti che il registro X viene utilizzato come puntatore corrente per puntare la ricerca all'elemento corrente della stringa. Per la ricerca dell'elemento corrente della stringa viene naturalmente utilizzato l'indirizzamento indicizzato.

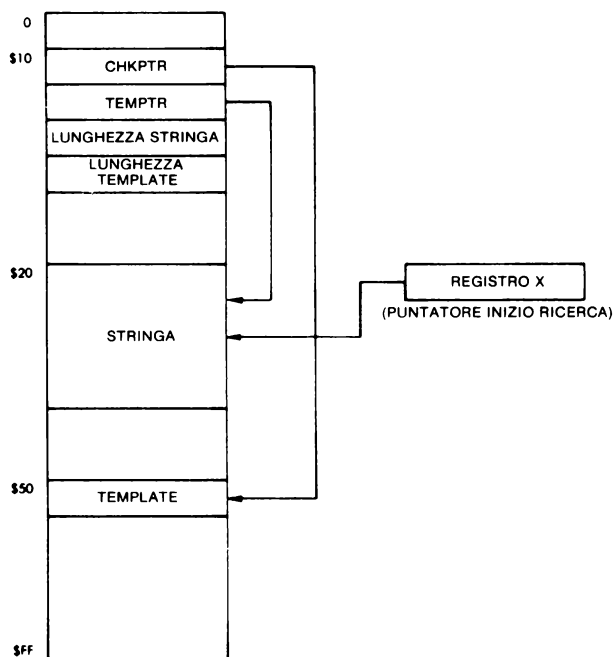



Figura 8.1: Ricerca di stringa: la memoria

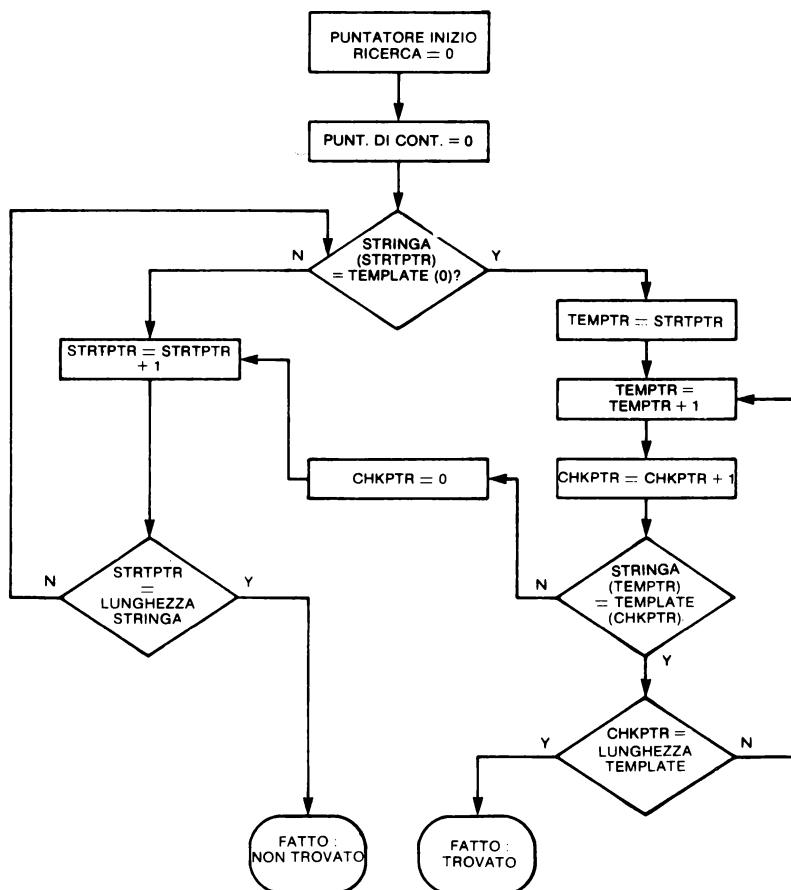



Figura 8.2: Diagramma di flusso del programma: ricerca di stringa

```
LINEA  #LOC  CODICE  LINEA

0002   0000          ; RICERCA DI STRINGA.
0003   0000          ; RICERCA LA LOCAZIONE NELLA STRINGA DI LUNGHEZZA
                     ; 'STRLEN'.
0004   0000          ; PARTENZA A 'STRING' DI UNA TEMPLATE DI
0005   0000          ; LUNGHEZZA 'TPTLEN' INIZIANTE A 'TEMPLT' E
0006   0000          ; RITORNO CON X = LOCAZIONE DI TEMPLATE
0007   0000          ; NELLA STRINGA SE TROVATA, OPPURE X =$ FF SE NON
                     ; TROVATA.
0008   0000          ;
0009   0000          STRING   =$20    ; PRIMA LOCAZIONE DELLA STRINGA.
0010   0000          TEMPLT   =$50    ; PRIMA LOCAZIONE DI TEMPLATE.
0011   0000                   *=$10
```

Figura 8-3. Programma per la Ricerca di Stringa (continua)

```
0012  0010              CHKPTR  *=*+1
0013  0011              TEMPTR  *=*+1
0014  0012              STRLEN  *=*+1  ; LUNGHEZZA DELLA STRINGA.
0015  0013              TPTLEN  *=*+1  ; LUNGHEZZA DI TEMPLATE.
0016  0014                      *=*$200
0017  0200  A2 00               LDX  #0          ; RESET PUNTATORE INIZIO
                                                 ; RICERCA.
0018  0202  A5 50       NXTPOS  LDA  TEMPLT      ; IL PRIMO ELEMENTO DI
                                                 ; TEMPLATE È ...
0019  0204  D5 20               CMP  STRING, X   ; = ALL'ELEMENTO
                                                 ; CORRENTE DELLA STRINGA?
0020  0206  F0 08               BEQ  CHECK       ; SE SI CONTROLLA IL RESTO
0021  0208  EB          NXTSTR  INX              ; INCREMENTA CONTATORE
                                                 ; INIZIO RICERCA
0022  0209  E4 12               CPX  STRLEN      ; È UGUALE A LUNGHEZZA
                                                 ; STRINGA?
0023  020B  D0 F5               BNE  NEXPOS      ; NO, CONTROLLA
                                                 ; SUCCESSIVA POSIZIONE
                                                 ; STRINGA
0024  020D  A2 FF               LDX  #$FF        ; SI,PONI AD 1 L'INDICATORE
                                                 ; "NON TROVATO".
0025  020F  60                  RTS              ; RITORNO: CONTROLLATI
                                                 ; TUTTI I CARATT.
0026  0210  86 11       CHECK   STX  TEMPTR      ; PONI PUNTATORE
                                                 ; TEMPORANEO=.
0027  0212                                       ; PUNTATORE CORRENTE
                                                 ; DELLA STRINGA.
0028  0212  A9 00               LDA  #0
0029  0214  85 10               STA  CHKPTR      ; RESET PUNTATORE
                                                 ; TEMPLATE.
0030  0216  E6 11       CHKLP   INC  TEMPTR      ; INCREMENTA PUNTATORE
                                                 ; TEMPORANEO.
0031  0218  E6 10               INC  CHKPTR      ; INCREMENTA PUNTATORE
                                                 ; TEMPLATE.
0032  0214  A4 10               LDY  CHKPTR
0033  021C  C4 13               CPY  TPTLEN      ; È PUNTATORE TEMPLATE =
                                                 ; LUNGHEZZA TEMPLATE?
0034  021E  F0 0C               BEQ  FOUND       ; SE SI TEMPLATE È TROVATA
0035  0220  B9 50 00            LDA  TEMPLT, Y   ; CARICA ELEMENTO
                                                 ; TEMPLATE
0036  0223  A4 11               LDY  TEMPTR
0037  0225  D9 20 00            CMP  STRING, Y   ; CONFRONTA COL
                                                 ; CARATTERE STRINGA
0038  0228  D0 DE               BNE  NXTSTR      ; SE NON TROVATO
                                                 ; CONTROLLA SUCCESSIVO
                                                 ; CARATTERE STRINGA.
0039  022A  F0 EA               BEQ  CHKLP       ; SE TROVATO CONTROLLA
                                                 ; CAR. SUCCESS
0040  022C  60          FOUND   RTS              ; FATTO
0041  022D                      END              ;
```

Figura 8-3. Programma per la Ricerca di Stringa

## SOMMARIO

In questo capitolo sono state presentate routine utilizzate comunemente che impiegano le combinazione di tecniche descritte nei capitoli precedenti. Queste dovrebbero consentire il progetto autonomo di programmi. Molte di queste routine impiegano una struttura dati speciale: la tabella. Esistono altre possibilità di strutturazione dei dati che verranno ora analizzate.

CAPITOLO 9

# STRUTTURE DEI DATI

## PARTE I - CONCETTI DI PROGETTO

### INTRODUZIONE

Il progetto di un programma comprende due compiti: *progetto dell'algoritmo e progetto delle strutture dati.* Nei programmi più semplici non vengono considerate strutture dati significative cosicchè il problema principale da superare per imparare la programmazione è l'apprendimento del progetto degli algoritmi e la loro codifica efficiente in un dato linguaggio di macchina. Questo è quanto è stato fatto fin'ora. Comunque il progetto di programmi più complessi richiede anche una comprensione delle strutture dati. Due strutture dati sono già state utilizzate nel corso del libro: la tabella e lo stack. Lo scopo di questo capitolo è di presentare altre strutture dati, più generali, che si può voler utilizzare. Questo capitolo è completamente indipendente dal microprocessore, od anche il calcolatore considerato. Questo è teorico e comprende l'organizzazione logica dei dati nel sistema. Esistono libri specializzati sull'argomento delle strutture dati come pure esistono libri specializzati sulla moltiplicazione efficiente, divisione ed altri algoritmi consueti. Questo capitolo è stato perciò introdotto per completezza ma sarà limitato all'essenziale. Esso non pretende di essere completo. Verranno ora analizzate le strutture dati più comuni.

### PUNTATORI

Un puntatore è un numero utilizzato per designare la locazione corrente del dato. Ciascun puntatore è un indirizzo. Comunque ciascun indirizzo non è necessariamente chiamato un puntatore. Un indirizzo è un puntatore solo se esso punta ad alcuni tipi di dati ovvero ad informazioni strutturate. È già stato incontrato un puntatore tipico: il puntatore dello stack che punta alla sommità dello stack (od anche immediatamente sopra la sommità dello stack). Si vedrà che lo stack ha una struttura dati comune chiamata una struttura LIFO.

Come altro esempio, quando si utilizza l'indirizzamento indiretto, l'indirizzamento indiretto è sempre un puntatore ai dati che si desidera recuperare.

**Esercizio 9-1:** *All'indirizzo 15 della memoria c'è un puntatore alla tabella T. La tabella T inizia all'indirizzo 500. Quali sono i contenuti effettivi del puntatore a T?*

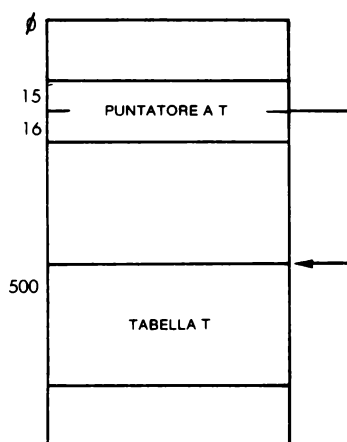



Figura 9.1: Un puntatore di indirizzamento

## LISTE

Quasi tutte le strutture dati sono organizzate come liste di vario tipo.

### Liste Sequenziali

Una lista sequenziale, o tabella, o blocco, è probabilmente la struttura dati più semplice ed una di quelle già utilizzate. Le tabelle sono normalmente ordinate in funzione di un criterio specifico, come per esempio, l'ordine alfabetico oppure quello numerico. È quindi facile recuperare un elemento in una tabella utilizzando, per esempio, l'indirizzamento indicizzato, come si è già fatto. Normalmente un blocco fa riferimento ad un gruppo di dati che hanno limiti definiti ma i cui contenuti non sono ordinati. Esso può contenere, per esempio, una stringa di caratteri. Oppure può essere un settore di un disco. In questi casi può non essere facile accedere ad elementi casuali del blocco.

Per facilitare la ricerca di blocchi di informazione sono utilizzati i direttori.

## Direttori

Un direttorio è una lista di tabelle o blocchi. Per esempio il sistema file utilizzerà normalmente una struttura a direttorio. Come semplice esempio il direttorio principale del sistema può comprendere una lista di nomi di utenti. Questo è illustrato in Figura 9-2. L'ingresso per l'utente "Giovanni" punta al direttorio del file di Giovanni. Il direttorio del file è una tabella che contiene i nomi di tutti i file di Giovanni e la loro locazione. Questa è, a sua volta, una tabella di puntatori. In questo caso si è quindi considerato un direttorio a due livelli. Un sistema a direttorio flessibile consentirà di comprendere direttori intermedi, a seconda della convenienza dell'utente.

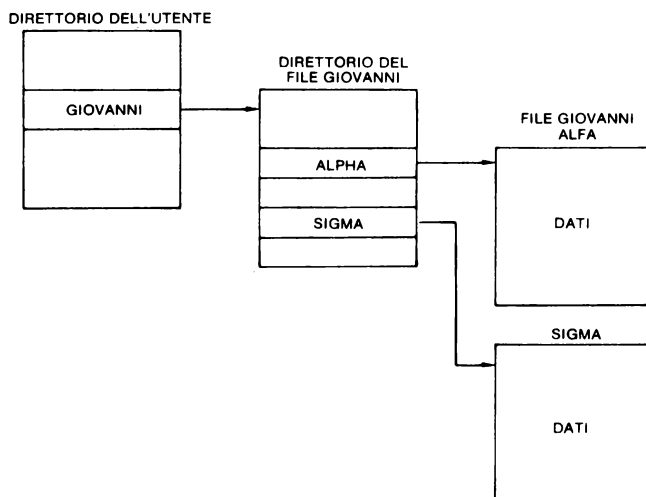



Figura 9.2: Una struttura a direttorio

## Lista Collegata

In un sistema ci sono spesso blocchi di informazioni che rappresentano dati, oppure eventi, oppure altre strutture, che non possono essere facilmente manipolate. Se questi potessero essere facilmente manipolati verrebbero probabilmente assemblati in una tabella per avere la possibilità di scelta o strutturazione. Il problema consiste nel fatto che si

desidera lasciarli dove sono pur stabilendo un ordinamento tra di essi come primo, secondo, terzo, quarto blocco. Per risolvere questo problema verrà impiegata una lista collegata. Il concetto di una lista collegata è illustrato dalla Figura 9-3. Nell'illustrazione si vede che un puntatore della lista, chiamato PRIMOBLOCCO punta all'inizio del primo blocco. Una locazione del Blocco 1, per esempio la prima o l'ultima parola di questo, contiene il puntatore al Blocco 2, chiamato PTR1. Il processo è quindi ripetuto per il Blocco 2 e per il Blocco 3. Poichè il blocco 3 è l'ultimo ingresso della lista, PTR3, per convenzione contiene uno speciale valore "nil" che punta a se stesso e che può essere rivelato alla fine della lista. Questa struttura è economica poichè essa richiede solo pochi puntatori (uno per blocco) e consente all'utente di non avere il movimento fisico dei blocchi nella memoria.

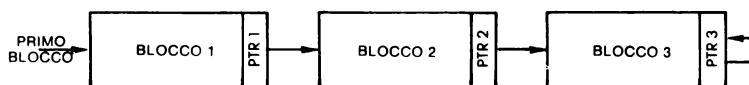



Figura 9.3: Una lista collegata

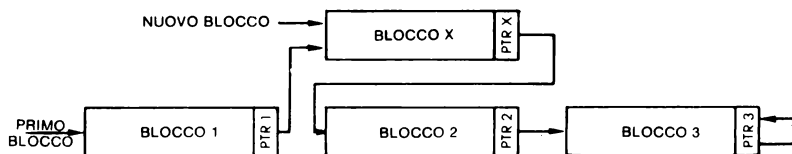



Figura 9.4: Inserzione di un nuovo blocco

Si esamini, per esempio, come può essere inserito un nuovo blocco. Questo è illustrato dalla Figura 9-4. Si assuma che il nuovo blocco sia all'indirizzo NUOVOBLOCCO e debba essere inserito tra il Blocco 1 ed il Blocco 2. Il puntatore PTR1 viene semplicemente cambiato al valore NUOVOBLOCCO cosicchè esso punta al Blocco X. PTRX conterrà il valore precedente di PTR1, cioè esso punterà al Blocco 2. Gli altri contatori della struttura rimangono invariati. Si può vedere che l'inserzione di un nuovo blocco ha richiesto semplicemente l'aggiornamento di due puntatori della struttura. Questo è chiaramente efficiente.

**Esercizio 9-2:** *Si tracci un diagramma che mostri come il Blocco 2 potrebbe essere rimosso da questa struttura.*

## Coda (Queue)

Una coda è formalmente chiamata una lista FIFO ovvero first-in-first-out. Una coda è illustrata in Figura 9-5. Per chiarire il diagramma si può assumere per esempio che il blocco di sinistra sia una routine di servizio per un dispositivo d'uscita, come una stampante. I blocchi che compaiono sulla destra sono quelli richiesti dai vari programmi o routine per stampare caratteri. L'ordine in cui essi saranno asserviti è l'ordine stabilito dalla coda di servizio. Si può vedere che l'evento successivo che ottiene servizio è il Blocco 1 poi il Blocco 2 e quindi il Blocco 3. In una coda si conviene che qualunque elemento arrivato successivamente sarà inserito alla fine di essa in questo caso sarà inserito dopo PTR3. Questo garantisce che il primo blocco inserito nella coda

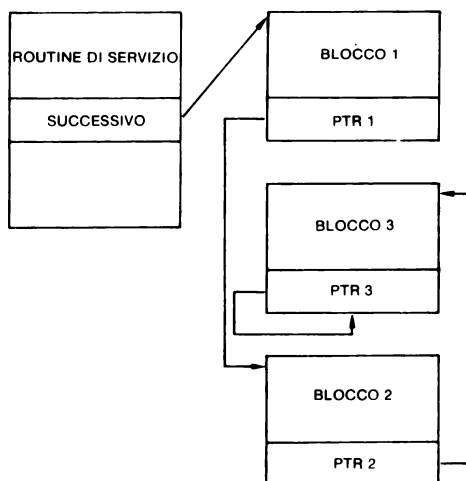



Figura 9.5: Una coda

sarà il primo ad essere asservito. E abbastanza comune in un sistema a calcolatore avere code di attesa per un certo numero di eventi ogni volta che si deve attendere una risorsa scarsa come il processore o qualche dispositivo d'ingresso/uscita.

Sono state sviluppate diversi tipi di liste per facilitare tipi specifici di accesso oppure inserzione o cancellazione alla lista stessa. Si esamineranno alcuni dei tipi di liste collegate utilizzati più frequentemente.

## Stack

La struttura stack è già stata studiata in dettaglio nel corso del libro. Essa è una struttura last-in-first-out (LIFO). L'ultimo elemento depositato alla sua sommità è il primo ad essere rimosso. Uno stack può essere realizzato mediante un blocco a scelta ovvero anche mediante una lista. Poichè la maggior parte degli stack dei microprocessori sono utilizzati per eventi ad alta velocità, come subroutine od interrupt, per lo stack viene normalmente utilizzato un blocco continuo piuttosto che una lista collegata.

## Confronto tra lista collegata e Blocco

Analogamente la coda potrebbe essere realizzata con un blocco di locazioni riservate. Il vantaggio di utilizzare un blocco continuo è il recupero veloce e l'eliminazione dei puntatori. Lo svantaggio consiste nel fatto che è normalmente necessario dedicare un blocco abbastanza largo per comprendere la dimensione del caso peggiore della struttura. Inoltre è difficoltoso od addirittura impraticabile inserire o rimuovere elementi dall'interno del blocco. Poichè la memoria è tradizionalmente una risorsa scarsa i blocchi vengono tradizionalmente riservati alle strutture di dimensione fissa ovvero alle strutture che richiedono la massima velocità di recupero, come lo stack.

## Lista Circolare

La lista circolare viene comunemente chiamata "round robin". Una lista circolare è una lista collegata dove l'ultimo punto rientra al primo. Questo è illustrato in Figura 9-6. Nel caso di una lista circolare viene

Figura 9.6: Il Round Robin è una lista circolare

spesso impiegato un puntatore al blocco attuale. Nel caso di eventi o programmi, attesa di servizio, il puntatore all'evento attuale sarà mosso di una posizione a sinistra oppure a destra, ad ogni volta.
Un round-robin corrisponde normalmente alla struttura dove tutti i blocchi sono assunti avere la stessa priorità. Comunque una lista circo-

lare può essere anche utilizzata come un sottocaso di altre strutture semplicemente per facilitare il recupero del primo blocco dopo l'ultimo, quando si sta eseguendo una ricerca.

Come esempio di lista circolare un programma di registrazione nor malmente opera in modo round-robin interrogando tutte le periferiche e ritornando indietro alla prima.

## Alberi

Ogni volta che esiste una relazione logica tra tutti gli elementi di una struttura (questa è chiamata normalmente una sintassi), può essere utilizzata una struttura ad albero. Un esempio semplice di una struttura ad albero è un albero discendente oppure un albero genealogico. Questo è illustrato in Figura 9-7. Si può vedere che Smith ha due bambini: un figlio Robert ed una figlia Jane. Jane, a sua volta, ha tre bambini: Liz, Tom e Phil. Tom a sua volta ha due bambini: Max e Chris. Invece Robert, riportato a sinistra dell'illustrazione non ha discendenti.

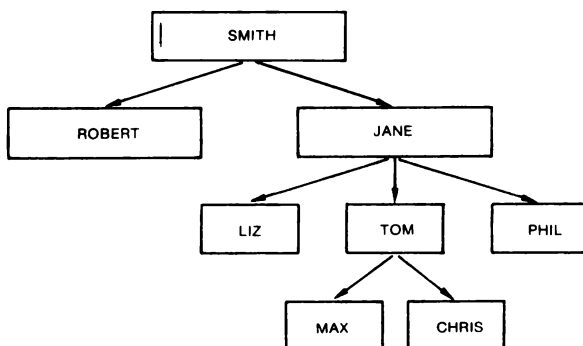



Figura 9.7: Albero genealogico

Questo è un albero strutturato. Si è già incontrato un esempio di un albero semplice in Figura 9-2. La struttura a direttorio è un albero a due livelli. Gli alberi sono utilizzati vantaggiosamente ogni volta che gli elementi possono essere classificati secondo una struttura prefissata. Questo facilita l'inserzione ed il recupero. Inoltre essi possono stabilire gruppi di informazione in un modo strutturato. Questo può essere richiesto per un'elaborazione ulteriore, come nel progetto di un compilatore od interprete.

## Liste Doppiamente Collegate

Collegamenti addizionali possono essere stabiliti tra gli elementi di una lista. L'esempio più semplice è la lista doppiamente collegata. Questo è illustrato in Figura 9-8. Si può vedere che sussiste la sequenza usuale di collegamenti da sinistra a destra, più un'altra sequenza di collegamenti da destra a sinistra. Lo scopo è di consentire un facile recupero dell'elemento immediatamente precedente quello che sta per essere processato come pure di quello immediatamente dopo. Questo costituisce un ulteriore puntatore per il blocco.

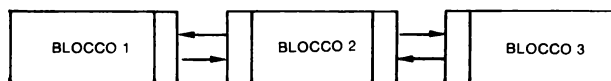



Figura 9.8: Lista doppiamente collegata

## RICERCA E CLASSIFICAZIONE

La ricerca e la classificazione degli elementi di una lista dipende direttamente dal tipo della struttura che è stata utilizzata per la lista. Molti algoritmi di ricerca sono stati sviluppati per le strutture dati utilizzate più frequentemente. Si è già utilizzato l'indirizzamento indicizzato. Questo è possibile ogni volta che gli elementi di una tabella sono ordinati in funzione di un criterio noto. Tali elementi possono poi essere recuperati mediante i loro numeri.

La *ricerca sequenziale* fa riferimento alla scansione lineare di un intero blocco. Questo è chiaramente inefficiente ma può essere utilizzato quando non è disponibile una tecnica migliore per mancanza di ordinamento degli elementi. La *ricerca binaria o logaritmica* serve a trovare un elemento in una lista classificata dividendo a metà l'intervallo di ricerca a ogni fase. Assumendo, per esempio, che si stia cercando una lista alfabetica si può iniziare, per esempio, a metà della tabella e determinare se il nome che si sta cercando è prima o dopo di questo punto. Se è dopo questo punto si eliminerà la prima metà della tabella e si osserverà la seconda metà. Si confronterà ancora questo ingresso con quello che si sta osservando e si restringerà la ricerca ad una delle ulteriori metà, eccetera. La lunghezza massima della ricerca è garantita essere $\log_2 n$ dove n è il numero di elementi della tabella.

Esistono molte altre tecniche di ricerca.

## SOMMARIO

Questo capitolo si è proposto solo una breve presentazione delle strutture dati usuali che possono essere utilizzate da un programmatore. Sebbene le strutture dati più comuni sono state razionalizzate in tipi cui è stato assegnato un nome, l'organizzazione globale dei dati in un sistema complesso può utilizzare qualsiasi combinazione di questi oppure richiedere al programmatore di inventare strutture più appropriate. L'insieme di possibilità è limitato solo dall'immaginazione del programmatore. Analogamente un numero di ben note tecniche di ricerca e classificazione sono state sviluppate per accoppiarsi con le usuali strutture dati. Lo scopo di questo libro è una descrizione concettuale. I contenuti di questo libro sono intesi a sottolineare l'importanza del progetto di strutture dati appropriate per la manipolazione dei dati e per fornire strumenti appropriati a questo effetto.

CAPITOLO 9
# STRUTTURE DEI DATI

## PARTE II - ESEMPI DI PROGETTO

### INTRODUZIONE

Verranno qui presentati degli esempi di progetto reali per strutture dati tipiche: tabelle, linked list, alberi di classificazione. Si eseguiranno i programmi per queste strutture, degli algoritmi reali di classificazione, ricerca ed inserzione. Verranno inoltre descritte delle tecniche aggiuntive avanzate quali hashing e merging.

Il lettore interessato a queste tecniche di programmazione avanzata viene incoraggiato ad analizzare i dettagli dei programmi di seguito presentati. Invece i programmatori meno esperti potranno inizialmente tralasciare questo capitolo, per poi rivederlo in una fase successiva.

Una buona comprensione dei concetti presentati nella prima parte di questo capitolo è indispensabile per seguire gli esempi di progetto. Inoltre i programmi impiegano i modi di indirizzamento del 6502, integrando molti dei concetti e delle tecniche presentate nei capitoli precedenti.

Verranno ora introdotte quattro strutture: una lista semplice, una lista alfabetica, una linked list con direttori ed un albero. Per ogni struttura verranno sviluppati tre programmi: ricerca, ingresso e cancellazione.

Inoltre verranno descritti separatamente alla fine del capitolo tre algoritmi specializzati: hashing, bubble-sort e merging.

### RAPPRESENTAZIONE DEI DATI DI UNA LISTA

La lista semplice e la lista alfabetica utilizzano una rappresentazione comune per ogni elemento della lista:

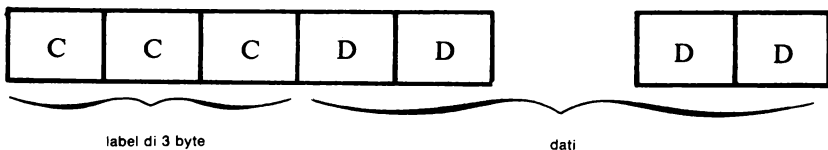
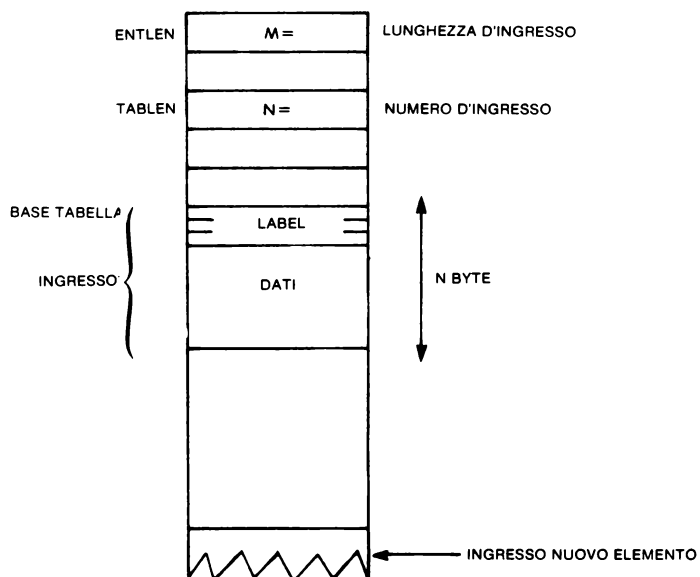



Figura 9.9: La struttura della tabella

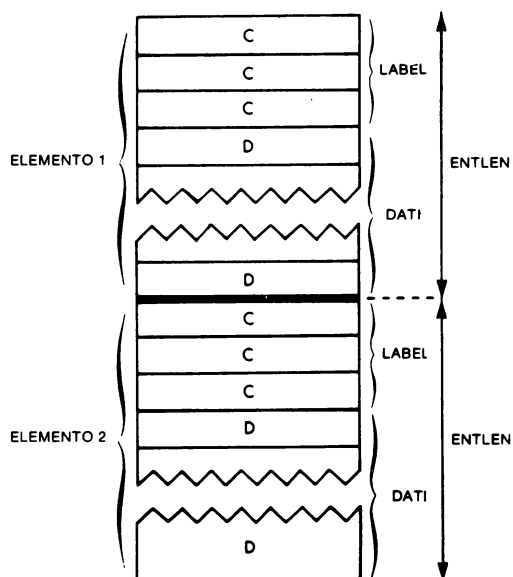



Figura 9.10: Ingressi tipici della lista in memoria

Ogni elemento, o "ingresso" comprende una label di tre byte ed un blocco i di n bye di dati con n tra 1 e 253. Quindi ogni ingresso impiega almeno una pagina (256 byte). All'interno di ogni lista, tutti gli elementi hanno la stessa lunghezza (Vedere Fig. 9-10). I programmi che operano su queste due semplici liste impiegano alcune convenzioni comuni sulle variabili:

ENTLEN è la lunghezza di un elemento. Per esempio, se ogni elemento ha 10 byte di dati, ENTLEN = 3 + 10 = 13 byte
TABASE è la base della lista o tabella nella memoria
POINTR è il puntatore all'elemento corrente
OBJECT è l'ingresso corrente da inserire o cancellare
TABLEN è il numero di ingressi

Si assume che tutte le label siano distinte.

## UNA LISTA SEMPLICE

La lista semplice è organizzata come tabella di n elementi. Gli elementi non sono classificati (vedere Fig. 9-11). Durante la ricerca occorre esplorare la lista fino a trovare l'ingresso oppure arrivare alla fine della

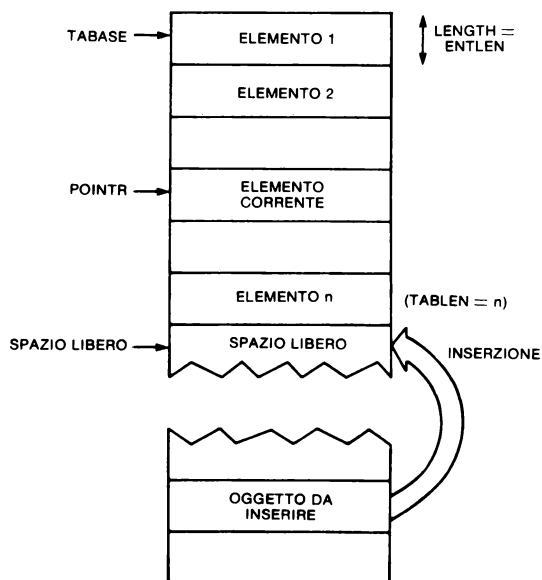



Figura 9.11: La lista semplice

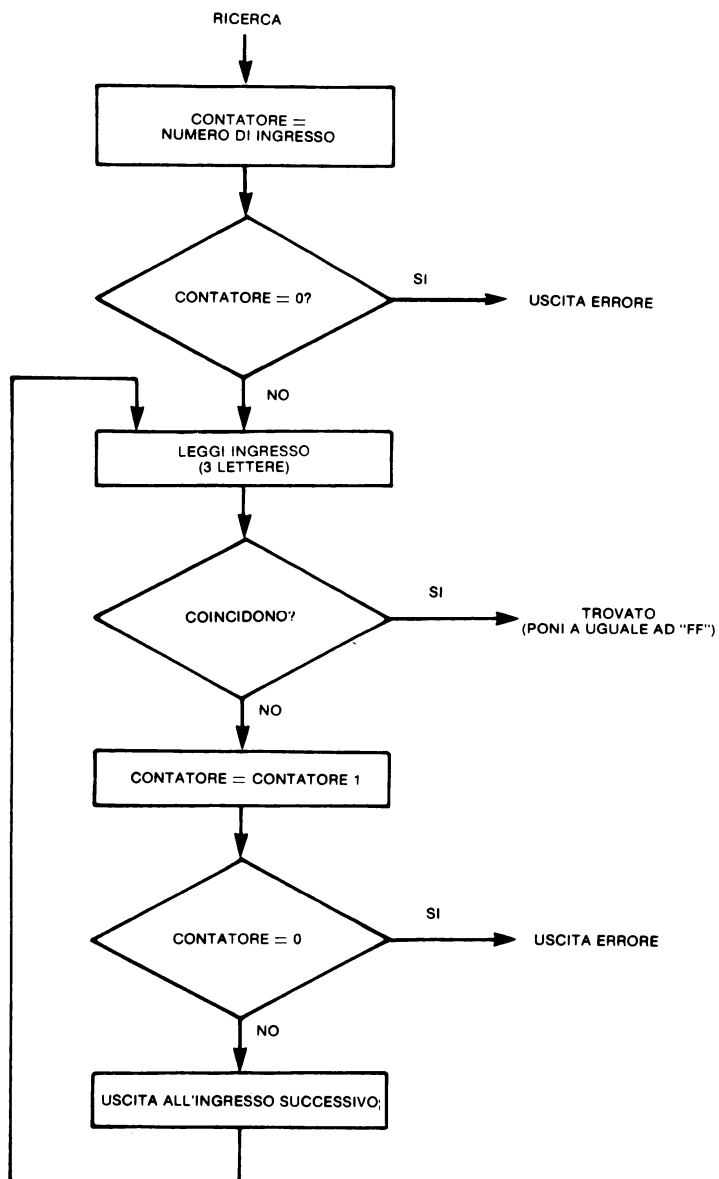



Figura 9.12: Diagramma di flusso della ricerca in una tabella

lista. Durante l'inserzione vengono aggiunti n nuovi ingressi a quelli esistenti. Quando viene cancellato un ingresso, gli ingressi contenuti in locazioni di memoria che precedono, se presenti, verranno fatti scorrere in avanti per la continuità della tabella.

### Ricerca

Viene utilizzata una tecnica di ricerca seriale. Ogni campo della label dell'ingresso è confrontato passo a passo, con la label di OBJECT, lettera per lettera.

Il puntatore corrente POINTR viene inizializzato al valore di TABASE.

Il registro indice X viene inizializzato al numero di ingressi contenuti nella lista (memorizzato in TABLEN).

La ricerca procede in modo ovvio ed il relativo diagramma di flusso viene rappresentato in Fig. 9-12. La Fig. 9-16, alla fine del capitolo, riporta il programma. (Programma "SEARCH").

### Inserzione di elemento

Inserendo un nuovo elemento, viene utilizzato il primo blocco di memoria disponibile di (ENTLEN) byte alla fine della lista. (Vedere Fig. 9-11).

Inizialmente il programma controlla che il nuovo ingresso non sia già nella lista (in questo esempio si assume che tutte le label siano distinte). Se non è già nella lista viene incrementata la lunghezza della lista TABLEN e si trasferisce OBJECT alla fine della lista. La Fig. 9-13 mostra il diagramma di flusso corrispondente.

La Fig. 9-16, alla fine del capitolo, riporta il programma. Esso si chiama "NEW" e risiede alle locazioni di memoria da 0636 a 0659.

### Cancellazione di elemento

Per cancellare un elemento dalla lista, è sufficiente trasferire di una posizione tutti gli elementi che lo seguono ad un indirizzo più elevato. La lunghezza della lista viene decrementata. Questo procedimento viene illustrato in Fig. 9-14.

Il programma corrispondente è immediato ed è riportato in Fig. 9-16. Esso è denominato "DELETE" e risiede agli indirizzi di memoria da 0659 a 0686. La Fig. 9-15 riporta il diagramma di flusso.

La locazione di memoria TEMPTR viene utilizzata come puntatore temporaneo all'elemento da trasferire.

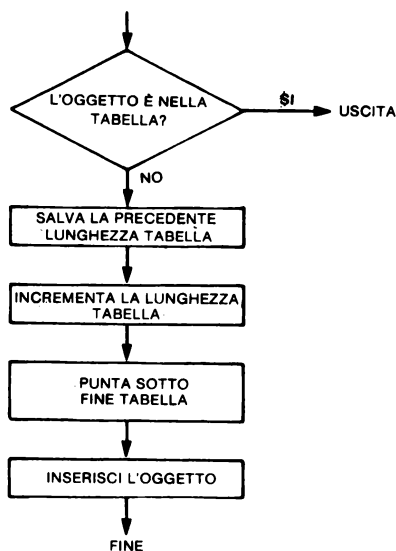



Figura 9.13: Diagramma di flusso di inserzione in tabella

Il registro indice Y contiene la lunghezza di un elemento della lista ed è impiegato per i trasferimenti automatici di blocchi di dati. Si noti che

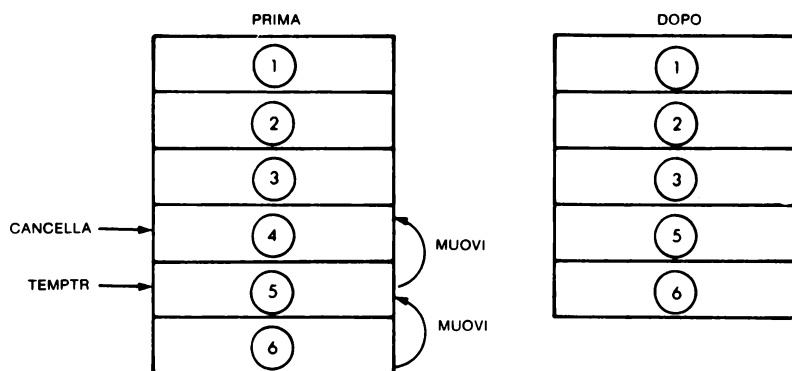



Figura 9.14: Cancellazione di un ingresso (lista semplice)

viene utilizzata la tecnica di indirizzamento indiretto indicizzato:

```
(0672)    LOOPE    DEY
                   LDA    (TEMPTR), Y
                   STA    (POINTR),Y
                   CPY    # 0
                   BNE    LOOPE
```

Durante i trasferimenti POINTR punta sempre al "buco" della lista, cioè alla destinazione del trasferimento del blocco successivo.

Il flag Z viene utilizzato per indicare una cancellazione sull'uscita.

## LISTA ALFABETICA

La lista alfabetica, o "tabella", rispetto a quella precedente, conserva tutti i suoi elementi classificati in ordine alfabetico. Questo consente all'utente una tecnica di ricerca più veloce rispetto alla tecnica lineare. In questo caso viene utilizzata una ricerca binaria.

### Ricerca

L'algoritmo di ricerca è quello classico della ricerca binaria. Si ricorda che questa tecnica è essenzialmente analoga a quella impiegata per trovare un nome in un elenco telefonico. Normalmente si parte a metà del libro e quindi, in dipendenza dell'ingresso trovato, si procede in avanti o indietro alla ricerca del valore desiderato. Questo metodo è veloce e relativamente semplice da realizzare.

Il diagramma di flusso della ricerca binaria è riportato in Fig. 9-17 ed il programma in Fig. 9-22.

La lista conserva gli elementi in ordine alfabetico e li ricerca impiegando una ricerca binaria o "logaritmica". La Fig. 9-18 riporta un esempio.

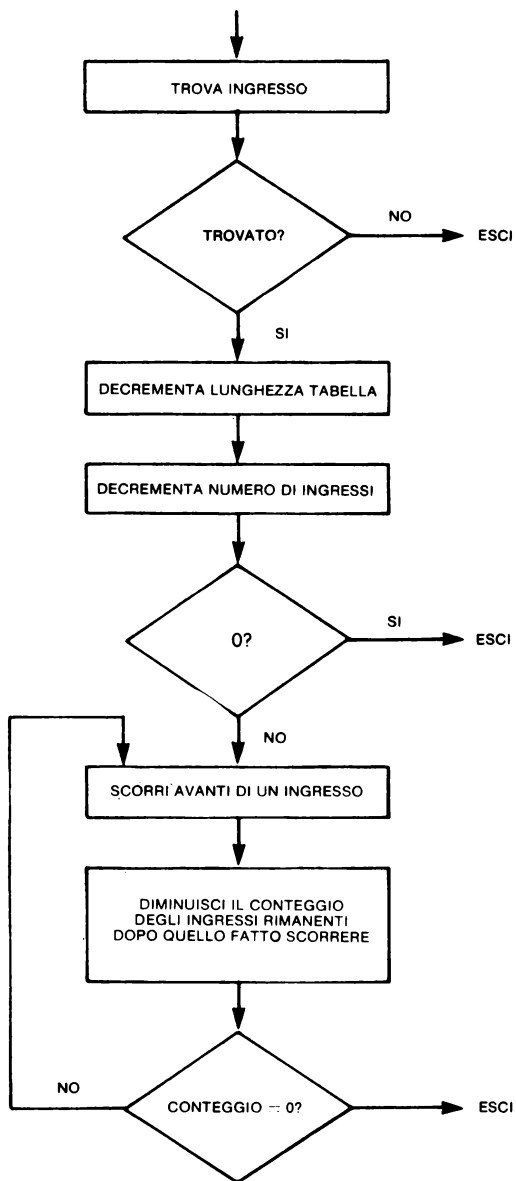



Figura 9.15: Diagramma di flusso di cancellazione in tabella

```
LINEA  #LOC CODICE    LINEA

0002   0000           ; TABASE =$ 10
0003   0000           ; POINTR =$ 12
0004   0000           ; TABLEN =$ 14
0005   0000           ; OBJECT =$ 15
0006   0000           ; ENTLEN =$ 17
0007   0000           ; TEMPTR =$ 18
0008   0000           ;
0009   0000                   *=$ 600
0010   0600                   ;
0011   0600   A5 10   SEARCH  LDA  TABASE       ; INIZIALIZZA PUNTATORE.
0012   0602   B5 12           STA  POINTR
0013   0604   A5 11           LDA  TABASE + 1
0014   0606   B5 13           STA  POINTR + 1
0015   060B   A6 14           LBX  TABLEN       ; IMMAGAZZINA TABLEN
                                                ; COME VARIABILE.
0016   060A   F0 29           BEQ  OUT          ; CONTROLLA SE TABELLA È 0.
0017   060C   A0 00   ENTRY   LBY  B0           ; CONFRONTA LE PRIME
                                                ; LETTERE.
0018   060E   B1 15           LBA  (OBJECT), Y
0019   0610   B1 12           CMP  (POINTR), Y
0020   0612   B0 0E           BNE  NOGOOD
0021   0614   CB              INY               ; CONFRONTA LE SECONDE
                                                ; LETTERE.
0022   0615   B1 15           LBA  (OBJECT), Y
0023   0617   B1 12           CMP  (POINTR), Y
0024   0619   B0 07           BNE  NOGOOD
0025   061B   CB              INY               ; CONFRONTA LE TERZE
                                                ; LETTERE.
0026   061C   B1 15           LBA  (OBJECT), Y
0027   061E   D1 12           CMP  (POINTR), Y
0028   0620   F0 11           BEQ  FOUND
0029   0622   CA      NOGOOD  DEX               ; VEDI QUANTI INGRESSI
                                                ; RIMANGONO.
0030   0623   F0 10           BEQ  OUT
0031   0625   A5 17           LDA  ENTLEN       ; SOMMA ENTLEN A POINTER.
0032   0627   18              CLC
0033   0628   65 12           ADC  POINTR
0034   062A   85 12           STA  POINTR
0035   062C   90 DE           BCC  ENTRY
0036   062E   A6 13           INC  POINTR + 1
0037   0630   4C 0C 06        JMP  ENTRY
0038   0633   A9 FF   FOUND   LDA  B$FF         ; SE TROVATO AZZERA
                                                ; IL FLAG 0.
0039   0635   60      OUT     RTS
0040   0636           ;
0041   0636           ;
0042   636            ;
0043   0636   20 00 06 NEW    JSR  SEARCH       ; VEDI SE L'OGGETTO È QUI.
0044   0639   D0 1D           BNE  OUTE
0045   063B   A6 14           LDX  TABLEN       ; CONTROLLA SE TABELLA È 0.
0046   063D   F0 0B           BEQ  INSERT
0047   063F   A5 12           LDA  POINTR       ; POINTER È ALL'ULTIMO
                                                ; INGRESSO.
0048   0641   18              CLC               ; .. DEVI TRASFERIRLO ALLA
                                                ; FINE DELLA TABELLA.
0049   0642   65 17           ADC  ENTLEN
0050   0644   B5 12           STA  POINTR
0051   0646   90 02           BCC  INSERT
0052   064B   E6 13           INC  POINTR + 1
```

Figura 9-16. Programmi della lista semplice: Ricerca, Inserzione, Cancellazione (continua)

```
0053   064A  E6 14      INSERT  INC   TABLEN        ; INCREMENTA LA
                                                    ; LUNGHEZZA DELLA
                                                    ; TABELLA.
0054   064C  A0 00              LDY   NO            ; TRASFERISCI L'OGGETTO
                                                    ; ALLA FINE DELLA TABELLA
0055   064E  A6 17              LDX   ENTLEN
0056   0650  B1 15      LOOP    LDA   (OBJECT), Y
0057   0652  91 12              STA   (POINTR), Y
0058   0654  C8                 INY
0059   0655  CA                 DEX
0060   0656  D0 F8              BNE   LOOP
0061   065B  60         OUTE    RTS                 ; Z AD 1 SE ERA FATTO
0062   0659             ;
0063   0659             ;
0064   0659             ;
0065   0659  20 00 06   DELETE  JSR   SEARCH        ; TROVA DOV'È L'OGGETTO.
0066   065C  F0 2D              BEQ   OUTS          ; ESCI SE NON TROVATO.
0067   065E  C6 14              DEC   TABLEN        ; INCREMENTA LUNGHEZZA
                                                    ; TABELLA.
0068   0660  CA                 DEX                 ; VEDI ORA QUANTI INGRESSI
                                                    ; SONO
0069   0061  F0 26              BEQ   DONE          ; .. DOPO AVERNE
                                                    ; CANCELLATO UNO.
0071   0063  A5 12      ADDEN   LDA   POINTR        ; SOMMA ENTLEN A POINTER
0072   0065  18                 CLC                 ;E ... MEMORIZZA A TEMP.
0073   0666  65 17              ADC   ENTLEN
0074   00668 85 18              STA   TEMPTR
0075   066A  A9 00              LDA   NO
       066C  65 13              ADC   POINTR + 1    ; SOMMA CARRY AL BYTE
                                                    ; ALTO.
0076   066E  85 19              STA   TEMPTR + 1
0077   0670  A4 17              LDY   ENTLEN
0078   0672  88         LOOPE   DEY
0079   0673  B1 18              LDA   (TEMPTR), Y   ; TRASFERISCI IN BASSO DI
                                                    ; UN INGRESSO DI MEMORIA.
0080   0675  91 12              STA   (POINTR), Y
0081   0677  C0 00              CPY   NO
0082   0679  D0 F7              BNE   LOOPE
0083   067B  CA                 DEX                 ; DECREMENTA
                                                    ; IL CONTATORE.
                                                     DEGLI INGRESSI
0084   067C  F0 0B              BEQ   DONE
0085   067E  A5 18              LDA   TEMPTR        ; TRASFERISCI TEMP
                                                    ; A POINTER.
0086   0680  85 12              STA   POINTR
0087   0682  A5 19              LDA   TEMPTR + 1
0088   0684  85 13              STA   POINTR + 1
0089   0686  4C 63 06           JMP   ADDEN
0090   0689  A9 FF      DONE    LDA   N$FF          ; AZZERA IL FLAG Z SE FATTO
0091   068B  60         OUTS    RTS
0092   068C             ;
0093   068C             ;
0094   068C                     END
ERRORS = 000 < 000 >

SYMBOL TABLE

SYMBOL  VALUE
ADDEN   0663   DELETE  0659   DONE    0689   ENTLEN  0017
ENTRY   060C   FOUND   0633   INSERT  064A   LOOP    0650
LOOPE   0672   NEW     0636   NOGOOD  0622   OBJECT  0015
OUT     0635   OUTE    0658   OUTS    068B   POINTR  0012
SEARCH  0600   TABASE  0010   TABLEN  0014   TEMPTR  0018

END OF ASSEMBLY
```

Figura 9.16: Programmi della lista semplice: Ricerca, Inserzione, Cancellazione

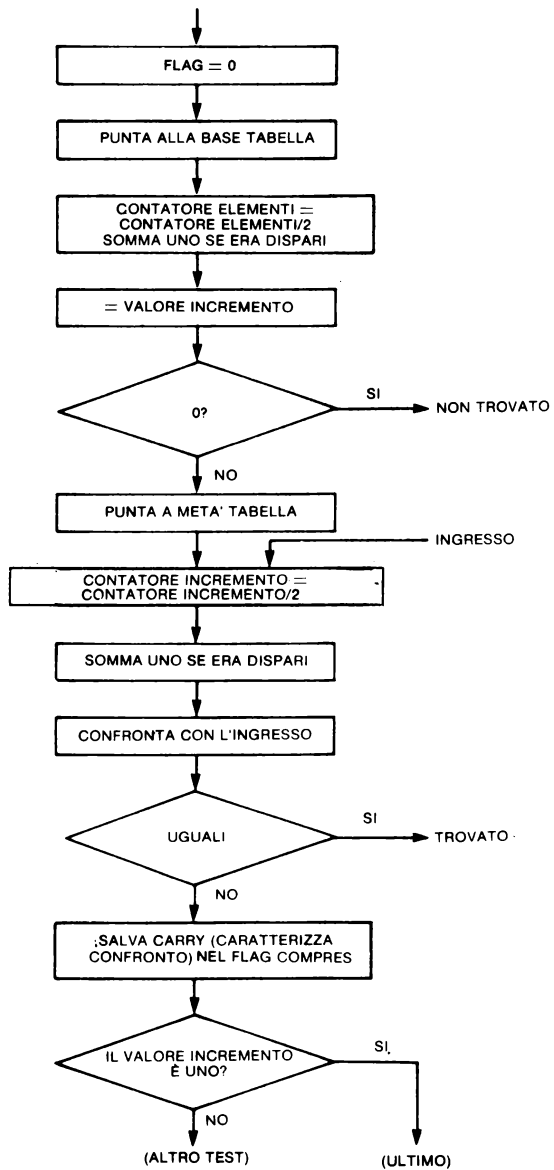



Figura 9.17: Diagramma di flusso della ricerca binaria (continua)

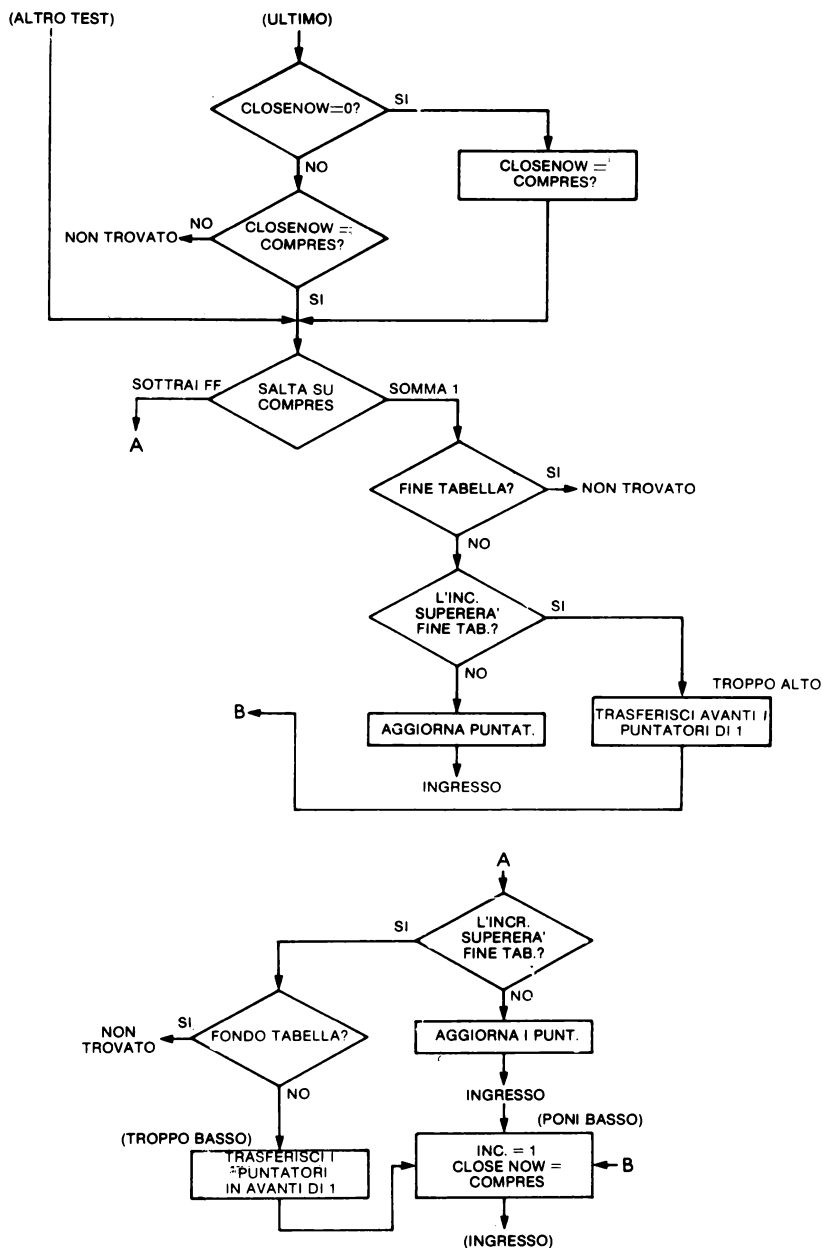



Figura 9-17: Diagramma di flusso della ricerca bianaria

Talvolta la ricerca è complicata dalla necessità di conservare la traccia di diverse condizioni. Il problema maggiore è quello di evitare la ricerca di un oggetto che non c'è. In tal caso l'ingresso entra con il valore alfabetico immediatamente più alto e più basso che dovrebbero essere controllati indefinitamente. Per evitare questo viene conservato un flag per il valore del carry dopo un confronto senza successo. Con il valore INCMNT. Quando il valore INCMNT, che mostra di quanto sarà incrementato il puntatore, raggiunge il valore "1", un altro flag "CLOSE" viene posto uguale al valore del flag CMPRS. Poichè tutti gli incrementi successivi saranno "1", se questo puntatore va diretto al punto dove dovrebbe essere l'oggetto, CMPRES non sarà più lungo o uguale di CLOSE e la ricerca terminerà. Questo caratterizza anche le abilitazioni della routine NEW per determinare se sono posizionati i puntatori logico e fisico, relativi a dove andrà l'oggetto.

Quindi se OBJECT cercato non si trova nella tabella, ed il puntatore corrente viene incrementato di uno, il flag CLOSE sarà posto uguale ad uno. Al passo successivo della routine, il risultato del confronto sarà opposto a quello precedente. I due flag non diventeranno uguali ed il programma terminerà indicando "non trovato".

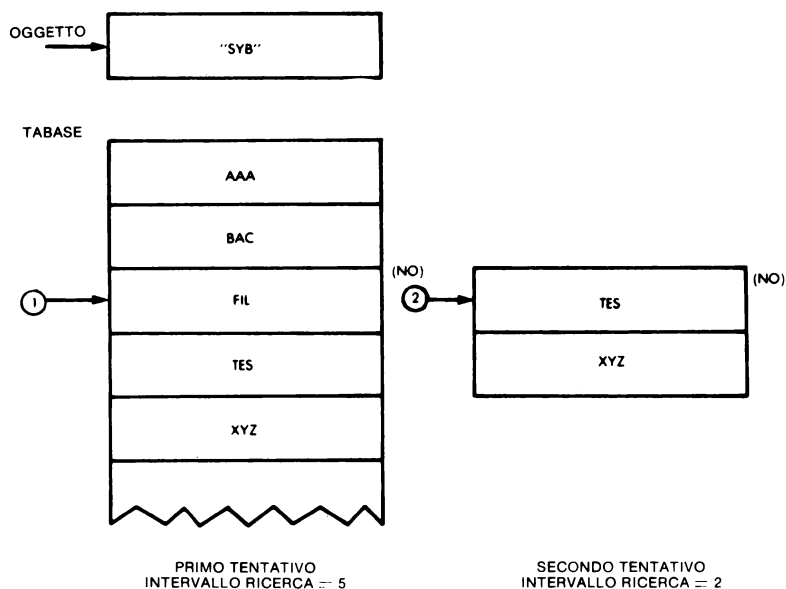



Figura 9.18: Una ricerca binaria

Un altro problema fondamentale è quello di evitare la possibilità di uscire fuori dalla fine della tabella quando si aggiunge o si sottrae il valore dell'incremento. Questo problema viene risolto eseguendo un test di "addizione" o di "sottrazione" utilizzando il puntatore logico ed il valore della lunghezza per determinare il numero effettivo di ingressi, piuttosto che utilizzare i puntatori fisici per determinare la loro posizione fisica effettiva.

Riassumendo, il programma impiega due flag per memorizzare l'informazione: CMPRES e CLOSE. Il flag CMPRES viene utilizzato per memorizzare il fatto che il carry era ancora "0" o "1" dopo l'ultimo confronto. Questo determina se l'elemento sotto test era più grande o più piccolo di quello a cui è confrontato. Ogni volta che il carry C è "1", l'ingresso è più piccolo dell'oggetto e CMPRES è posto ad "1". Ogni volta che il carry C è "0" l'ingresso è maggiore dell'oggetto e CMPRES sarà posto ad "FF".

Inoltre si noti che, quando il carry è 1, il puntatore corrente, punterà all'ingresso sotto OBJECT.

Il secondo flag impiegato dal programma è CLOSE. Questo flag è posto uguale a CMPRES quando l'incremento della ricerca INCMNT diviene uguale ad "1". Questo rivelerà il fatto che l'elemento non è stato trovato se CMPRES non è uguale a CLOSE la volta successiva.

Altre variabili utilizzate dal programma sono:

LOGPOS, che indica la posizione logica nella tabella (numero dell'elemento).

INCMNT, che rappresenta il valore di cui sarà incrementato o decrementato il puntatore corrente se non ha successo il confronto successivo.

TABLEN, come al solito, rappresenta la lunghezza totale della lista. LOGPOS ed INCMNT saranno confrontati con TABLEN per accertare che non vengano superati i limiti della lista.

La Fig. 9-22 rappresenta il programma chiamato "SEARCH'. Esso risiede alle locazioni di memoria da 0600 a 06E3 e merita di essere studiato con cura, in quanto è più complesso di quello del caso della ricerca lineare.

Una complicazione addizionale è dovuta al fatto che l'intervallo di ricerca può essere pari o dispari. Quando è pari occorre introdurre una correzione. Infatti, per esempio, esso non può puntare a metà di una lista di 4 elementi.

Quando questo è dispari, viene utilizzato un "artificio" per puntare all'elemento intermedio: si esegue la divisione per 2 accompagnata da uno scorrimento a destra. Il bit che va a cadere nel carry dopo l'istruzione LSR sarà "1" se l'intervallo era dispari. Esso viene semplicemente

aggiunto al puntatore:

```
(0615)      DIV      LSR      A           DIVIDE PER DUE
                     ADC      # 0         RIVELA IL CARRY
                     STA      LOGPOS      NUOVO PUNTATORE
```

OBJECT viene quindi confrontato con l'ingresso intermedio del nuovo intervallo di ricerca. Se il confronto da esito positivo, si esce dal programma, altrimenti ("NOGOOD") il carry è posto a "0" se OBJECT è minore dell'ingresso. Ogni volta che INCMNT diviene "1", il flag CLOSE (che è stato inizializzato a "0") viene controllato per vedere se è "1". In caso contrario quest'ultimo viene posto ad "1". Se era "1" si esegue un controllo per determinare se si è superata la locazione dove doveva essere OBJECT.

## Inserzione di Elemento

Per inserire un nuovo elemento è necessario eseguire una ricerca binaria. Se l'elemento in questione viene trovato nella tabella, non si deve eseguire l'inserzione. (In questo caso si assume che tutti gli elementi della tabella siano distinti). Se l'elemento non viene trovato nella tabella occorre procedere dalla sua inserzione. Il valore del flag CMPRES dopo la ricerca indica se questo elemento deve essere inserito immediatamente prima o dopo l'ultimo elemento che è stato confrontato. Tutti gli elementi successivi la nuova locazione dove deve essere posizionato l'elemento, vengono quindi trasferiti in avanti, di una posizione del blocco, consentendo l'inserzione del nuovo elemento.

La Fig. 9-19 illustra il processo di inserzione ed il programma corrispondente è riportato in Fig. 9-22.

Il programma è chiamato "NEW" e risiede nelle locazioni di memoria da 06E3 a 075E.

Si noti che, anche in questo caso, viene utilizzato l'indirizzamento indiretto indicizzato per i trasferimenti di blocco:

```
(072A)                 LDY      ENTLEN
         ANOTHR        DEY
                       LDA      (POINTR), Y
                       STA      (TEMP), Y
                       CPY      # 0
                       BNE      ANOTHR
```

Analogamente si procede alla locazione di memoria 0750.

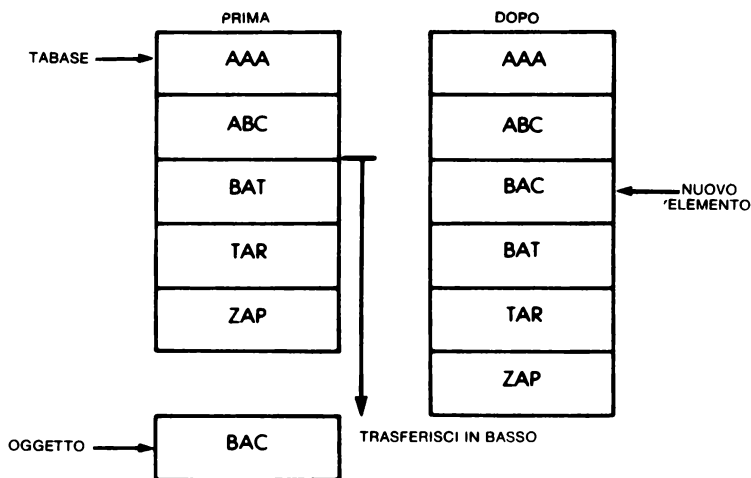



Figura 9.19: Inserzione: "BAC"

## Cancellazione di Elemento

Anche nel caso di cancellazione di un elemento, occorre utilizzare la tecnica di ricerca binaria per trovare l'oggetto. Se la ricerca da esito negativo, ovviamente la cancellazione non ha senso. Se invece, l'elemento viene trovato, tutti gli elementi successivi vengono mossi verso l'alto, di una posizione di blocco. La Fig. 9-20 mostra un esempio corrispondente e la Fig. 9-22 il programma relativo, mentre il diagramma di flusso appare in Fig. 9-21.

Esso è denominato "DELETE" e risiede agli indirizzi di memoria da 076F a 0799.

## LINKED LIST

Si assume che una linked list sia formata da tre caratteri alfanumerici per la label, seguiti da 1 a 256 byte di dati, seguiti da un puntatore a 2 byte che contiene l'indirizzo di partenza del nuovo ingresso ed, infine, seguito da un contrassegno di un byte. Ogni volta che questo contrassegno di un byte è posto ad "1", si previene che la routine di inserzione possa sostituire un nuovo ingresso al posto di quello esistente.

Inoltre un direttorio contiene un puntatore al primo ingresso per ogni lettera dell'alfabeto, in modo da facilitare la ricerca. Nel progrmma si assume che le label siano caratteri alfabetici ASCII. Alla fine della lista tutti i puntatori sono posti ad un valore NIL che è stato scelto, in questo caso, uguale alla base della tabella, in quanto questo valore non dovrebbe mai trovarsi all'interno della linked list.

Il programma di inserzione e di cancellazione esegue le manipolazioni ovvie sui puntatori. Questo impiega il flag INDEX per indicare se un puntatore sta puntando ad un oggetto proveniente da un ingresso precedente della lista o dalla tabella dei direttori. La Fig. 9-27 riporta i programmi corrispondenti, mentre la Fig. 9-23 mostra la struttura dati.

Un'applicazione di questa struttura dati potrebbe essere un elenco di indirizzi computerizzati, dove ogni persona è rappresentata da un solo codice di tre lettere (magari le comuni iniziali) ed il campo dei dati contiene un indirizzo semplificato, più il numero di telefono (fino a 250 caratteri).

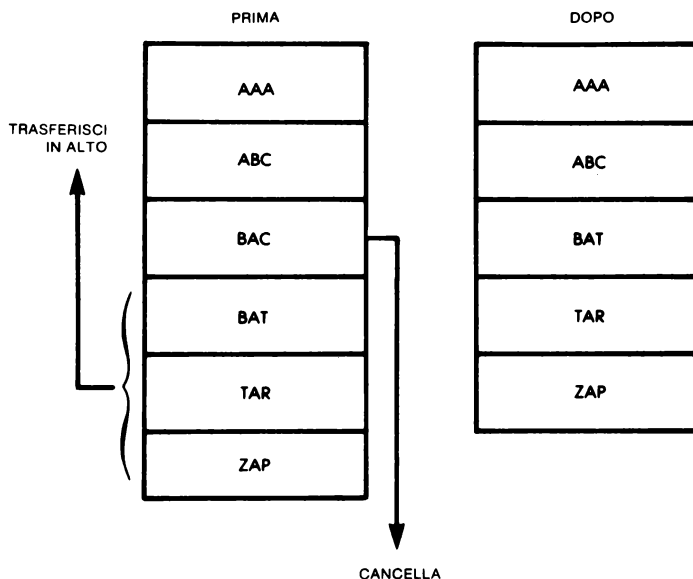



Figura 9.20: Cancellazione: "BAC"

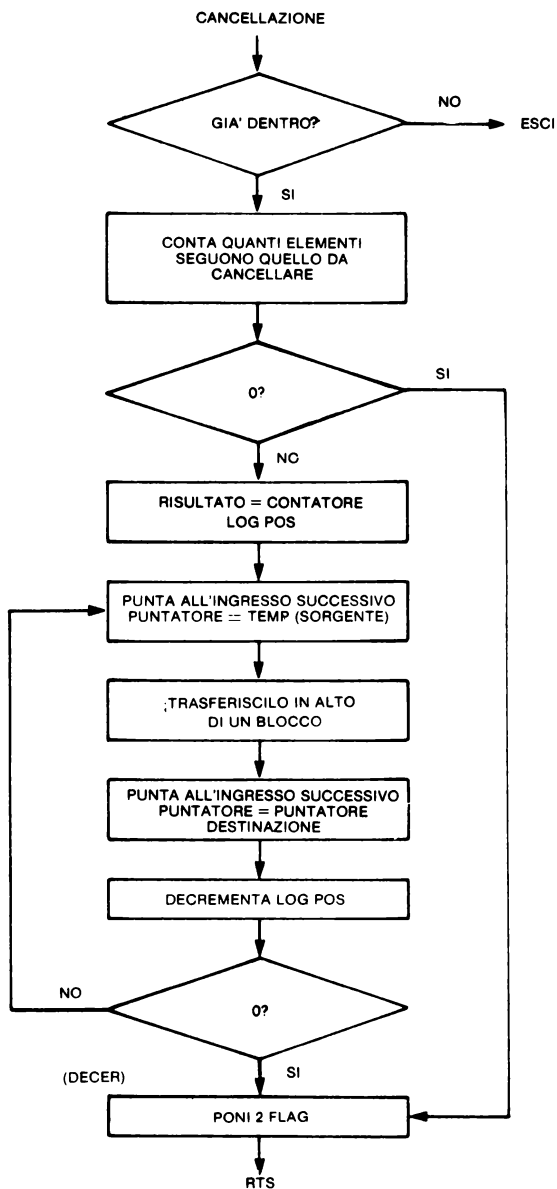



Figura 9.21: Diagramma di flusso di cancellazione (lista alfabetica)

```
LINEA  #LOC  CODICE     LINEA

0002   0000             CLOSE    =$ 10
0003   0000             CMPRES   =$ 11
0004   0000             TABASE   =$ 12
0005   0000             POINTR   =$ 14
0006   0000             TABLEN   =$ 16
0007   0000             LOGPOS   =$ 17
0008   0000             INCMNT   =$ 18
0009   0000             TEMP     =$ 19
0010   0000             ENTLEN   =$ 1B
0011   0000             OBJECT   =$ 1C
0012   0000             ;
0013   0000                      *=     $ 600
0014   0600             ;
0015   0600  A9 00      SEARCH   LDA    #0           ; FLAG ZERO.
0016   0602  85 10               STA    CLOSE
0017   0604  85 11               STA    CMPRES
0018   0606  A5 12               LDA    TABASE       ; INIZIALIZZA IL PUNTATORE.
0019   0608  85 14               STA    POINTR
0020   060A  A5 13               LDA    TABASE+1
0021   060C  85 15               STA    POINTR+1
0022   060E  A5 16               LDA    TABLEN       ; ACCETTA LUNGHEZZA
                                                     ; TABELLA
0023   0610  D0 03               BNE    DIV
0024   0612  4C E0 06            JMP    OUT
0025   0615  4A         DIV      LSR    A            ; DIVIDILA PER 2.
0026   0616  69 00               ADC    # 0          ; SOMMA AL PRIMO BIT.
0027   0618  85 17               STA    LOGPOS       ; MEMORIZZA COME
                                                     ; POSIZIONE LOGICA.
0028   061A  85 18               STA    INCMNT       ; MEMORIZZA COME VALORE
                                                     ; INCREMENTO.
0029   061C  A6 17               LDX    LOGPOS       ; MOLTIPLICA ENTLEN PER
                                                     ; LOGPOS.
0030   061E  CA                  DEX                 ; ... AGGIUNGENDO IL
                                                     ; RISULTATO AL PUNTATORE.
0031   061F  F0 0E               BEQ    ENTRY
0032   0621  A5 1B      LOOP     LDA    ENTLEN
0033   0623  18                  CLC
0034   0624  65 14               ADC    POINTR
0035   0626  85 14               STA    POINTR
0036   0628  90 02               BCC    LOOP
0037   062A  E6 15               INC    POINTR+1
0038   062C  CA         LOOP     DEX
0039   062D  D0 F2               BNE    LOOP
0040   062F  A5 18      ENTRY    LDA    INCMNT       ; DIVIDI IL VALORE
                                                     ; DELL'INCREMENTO PER 2.
0041   0631  4A                  LSR    A
0042   0632  69 00               ADC    # 0
0043   0634  85 18               STA    INCMNT
0044   0636  A0 00               LDY    # 0          ; CONFRONTA LE PRIME
                                                     ; LETTERE.
0045   0638  B1 1C               LDA    (OBJECT),Y
0046   063A  D1 14               CMP    (POINTR),Y
0047   063C  D0 11               BNE    NOGOOD
0048   063E  C8                  INY                 ; CONFRONTA LE SECONDE
                                                     ; LETTERE.
0049   063F  B1 1C               LDA    (OBJECT),Y
0050   0641  D1 14               CMP    (POINTR),Y
0051   0643  D0 0A               BNE    NOGOOD
```

Figura 9-22. Programma della Lista Alfabetica: Ricerca Binaria, Cancellazione, Inserzione (continua).

```
0052  0645  CB                 INY                ; CONFRONTA LE TERZE
                                                  ; LETTERE.
0053  0646  B1 1C              LDA   (OBJECT),Y
0054  0648  D1 14              CMP   (POINTR),Y
0055  064A  D0 03              BNE   NOGOOD
0056  064C  4C E2 06           JMP   FOUND
0057  064F  A0 FF     NOGOOD   LDY   #$FF         ; PONI IL FLAG DI CONFRONTO
                                                  ; RISULTATO
0058  0651  90 02              BCC   TESTS        ; SE OGGETTO < PUNTATORE:
                                                  ; C-0.
0059  0653  A0 01              LDY   # 1
0060  0655  84 11     TESTS    STY   CMPRES
0061  0657  A4 18              LDY   INCMNT       ; L'INCREMENTO HA VALORE 1?
0062  0659  88                 DEY
0063  065A  D0 10              BNE   NEXT
0064  065C  A5 10              LDA   CLOSE        ; CONTROLLA SE IL FLAG
                                                  ; "CLOSE" È 1.
0065  D65E  F0 08              BEQ   MAKCLO       ; SE NON È 1, VA A PORLO.
0066  0660  38        SEC
0067  0661  E5 11              SBC   CMPRES       ; VEDI SE SI È PASSATI DOVE
0068  0663  F0 07              BEQ   NEXT         ; ... DOVREBBE ESSERE
                                                  ; L'OGGETTO, MA NON C'È.
0069  0665  4C E0 06           JMP   OUT
0070  0668  A5 11     MAKCLO   LDA   CMPRES       ; POSIZIONA IL FLAG CLOSE
                                                  ; A CMPRES.
0071  066A  85 10              STA   CLOSE
0072  066C  24 11     NEXT     BIT   CMPRES
0073  066E  30 35              BMI   SUBIT
0074  0670  A5 16              LDA   TABLEN       ; CONTROLLA CHE
                                                  ; L'ADDIZIONE DI INCMNT
0075  0672  38                 SEC                ; ... VA OLTRE LA FINE TABELLA.
0076  0673  E5 17              SBC   LOGPOS
0077  0675  F0 69              BEQ   OUT          ; CONTROLLA SE SI È GIA'
                                                  ; A FINE TABELLA.
0078  0677  E5 18              SBC   INCMNT
0079  0679  90 1A              BCC   TOOHI
0080  067B  A6 18              LDX   INCMNT       ; SE VA BENE INCREMENTA
                                                  ; IL PUNTATORE DELLA
0081  067D  A5 1B     ADDER    LDA   ENTLEN       ; ...QUANTITA' CORRETTA.
0082  067F  18                 CLC
0083  0680  65 14              ADC   POINTR
0084  0682  85 14              STA   POINTR
0085  0684  90 02              BCC   AD1
0086  0686  E6 15              INC   POINTR+1
0087  0688  CA        AD1      DEX
0088  0689  D0 F2              BNE   ADDER
0089  068B  A5 17              LDA   LOGPOS       ; INCREMENTA LA POSIZIONE
                                                  ; LOGICA.
0090  068D  18                 CLC
0091  068E  65 18              ADC   INCMNT
0092  0690  85 17              STA   LOGPOS
0093  0692  4C 2F 06           JMP   ENTRY
0094  0695  E6 17     TOOHI    INC   LOGPOS       ; INCREMENTA LA POSIZIONE
                                                  ; LOGICA
0095  0697  A5 1B              LDA   ENTLEN       ; TRASFERISCI AVANTI
                                                  ; IL PUNTATORE DI UN
                                                  ; INGRESSO.
0096  0699  18                 CLC
0097  069A  65 14              ADC   POINTR
0098  069C  85 14              STA   POINTR
0099  069E  90 35              BCC   SETCLO
```

Figura 9-22. Programma della Lista Alfabetica: Ricerca Binaria, Cancellazione, Inserzione (continua).

```
0100   06A0  E6 15                 INC   POINTR+1
0101   06A2  4C D5 06              JMP   SETCLO
0102   06A5  A5 17       SUBIT     LDA   LOGPOS     ; VEDI SE INC VA OLTRE
                                                    ; IL FONDO
0103   06A7  38                    SEC              ; ... DELLA TABELLA.
0104   06AB  E5 18                 SBC   INCMNT
0105   06AA  F0 17                 BEQ   TOOLOW
0106   06AC  90 15                 BCC   TOOLOW
0107   06AE  85 17                 STA   LOGPOS     ; CONSERVA LA NUOVA
                                                    ; POSIZIONE LOGICA.
0108   06B0  A6 18                 LDX   INCMNT
0109   06B2  A5 14       SUBLOP    LDA   POINTR     ; SOTTRAI DAL CONTATORE
                                                    ; LA QUANTITA' CORRETTA.
0110   06B4  38                    SEC
0111   06B5  E5 1B                 SBC   ENTLEN
0112   06B7  85 14                 STA   POINTR
0113   06B9  B0 02                 BCS   SUB0
0114   06BB  C6 15                 DEC   POINTR+1
0115   06BD  CA          SUB0      DEX
0116   06BE  D0 F2                 BNE   SUBLOP
0117   06C0  4C 2F 06              JMP   ENTRY
0118   06C3  A6 17       TOOLOW    LDX   LOGPOS     ; VEDI SE POS È GIA' 1.
0119   06C5  CA                    DEX
0120   06C6  F0 18                 BEQ   OUT
0121   06C8  C6 17                 DEC   LOGPOS
0122   06CA  A5 14                 LDA   POINTR     ; SOTTRAI L'INGRESSO 1
                                                    ; DAL PUNTATORE.
0123   06CC  38                    SEC
0124   06CD  E5 1B                 SBC   ENTLEN
0125   06CF  85 14                 STA   POINTR
0126   06D1  B0 02                 BCS   SETCLO
0127   06D3  C6 15                 DEC   POINTR+1
0128   06D5  A9 01       SETCLO    LDA   # 1
0129   06D7  85 18                 STA   INCMNT
0130   06D9  A5 11                 LDA   CMPRES
0131   06DB  85 10                 STA   CLOSE
0132   06DD  4C 2F 06              JMP   ENTRY
0133   06E0  A2 FF       OUT       LDX   #$ FF      ; PONI Z AD 1 SE TROVATO.
0134   06E2  60          FOUND     RTS
0135   06E3                        ;
0136   06E3                        ;
0137   06E3                        ;
0138   06E3  20 00 06    NEW       JSR   SEARCH     ; VEDI SE L'OGGETTO È GIA'
                                                    ; QUI.
0139   06E6  F0 76                 BEQ   OUTE
0140   06EB  A5 16                 LDA   TABLEN     ; CONTROLLA SE TABELLA È 0.
0141   06EA  F0 62                 BEQ   INSERT
0142   06EC  24 11                 BIT   CMPRES     ; CONTROLLA RISULTATO
                                                    ; ULTIMO CONFRONTO.
0143   06EE  10 05                 BPL   LOSIDE
0144   06F0  C6 17                 DEC   LOGPOS     ; PONI LA POSIZIONE LOGICA
                                                    ; IN MODO CHE
0145   06F2  4C 00 07              JMP   SETUP      ; ... SUB OPERI
                                                    ; SUCCESSIVAMETE.
0146   06F5  A5 1B       LOSIDE    LDA   ENTLEN     ; PONI PUNTATORE PRIMA DI
                                                    ; DOVE
0147   06F7  18                    CLC              ; ... ANDRA' L'OGGETTO.
0148   06F8  65 14                 ADC   POINTR
0149   06FA  85 14                 STA   POINTR
0150   06FC  90 02                 BCC   SETUP
0151   06FE  E6 15                 INC   POINTR+1
0152   0700  A5 16       SETUP     LDA   TABLEN     ; VEDI QUANTI INGRESSI
```

Figura 9-22. Programma della Lista Alfabetica: Ricerca Binaria, Cancellazione, Inserzione (continua).

```
0153   0702  38               SEC            ; ... SONO DATI DOVE ANDRA'
                                             ; L'OGGETTO.
0154   0703  E5 17            SBC   LOGPOS
0155   0705  F0 47            BEQ   INSERT
0156   0707  AA               TAX
0157   0708  A8               TAY
0158   0709  88               DEY            ; GUARDA SE SI STA GIA'
                                             ; PUNTANDO
0159   070A  F0 0E            BEQ   SETEMP   ; ... ALL'ULTIMO INGRESSO.
0160   070C  A5 1B   UPLOOP   LDA   ENTLEN   ; MUOVI IL PUNTATORE
                                             ; ALL'ULTIMO INGRESSO.
0161   070E  18               CLC
0162   070F  65 14            ADC   POINTR
0163   0711  85 14            STA   POINTR
0164   0713  90 02            BCC   SET0
0165   0715  E6 15            INC   POINTR±1
0166   0717  88      SET0     DEY
0167   0718  D0 F2            BNE   UPLOOP
0168   071A  A5 14   SETEMP   LDA   POINTR   ; SOMMA ENTLEN A
                                             ; PUNTATORE E
0169   071C  18               CLC            ; ... MEMORIZZA A TEMP.
0170   071D  65 1B            ADC   ENTLEN
0171   071F  85 19            STA   TEMP
0172   0721  90 01            BCC   SET1
0173   0723  C8               INY            ; Y ERA GIA' 0.
0174   0724  98      SET 1    TYA
0175   0725  18               CLC
0176   0726  65 15            ADC   POINTR±1
0177   0728  85 1A            STA   TEMP+1
0178   072A  A4 1B   MOVER    LDY   ENTLEN   ; POSIZIONA Y PER LO
                                             ; SPOSTAMENTO.
0179   072C  88      ANOTHR   DEY
0180   072D  B1 14            LDA   (POINTR),Y ; MUOVI UN BYTE.
0181   072F  91 19            STA   (TEMP),Y
0182   0731  C0 00            CPY   # 0
0183   0733  D0 F7            BNE   ANOTHR
0184   0735  A5 14            LDA   POINTR   ; DECR. PUNTATORE E TEMP
0185   0732  38               SEC            ; ...DI ENTLEN.
0186   0738  E5 1B            SBC   ENTLEN
0187   073A  85 14            STA   POINTR
0188   073C  B0 02            BCS   M1
0189   073E  C6 15            DEC   POINTR±1
0190   0740  CA      M1       DEX
0191   0741  D0 D7            BNE   SETEMP
0192   0743  A5 1B            LDA   ENTLEN   ; TRASFERISCI INDIETRO
                                             ; IL PUNTATORE A
0193   0745  18               CLC            ; DOVE ANDRA' L'OGGETTO.
0194   0746  65 14            ADC   POINTR
0195   074B  85 14            STA   POINTR
0196   074A  90 02            BCC   INSERT
0197   074C  E6 15            INC   POINTR+1
0198   074E  40 00   INSERT   LDY   # 0
0199   0750  A6 1B            LDX   ENTLEN   ; TRASFERISCI L'OGGETTO
                                             ; NELLA TABELLA.
0200   0752  B1 1C   INNER    LDA   (OBJECT),Y
0201   0754  91 14            STA   (POINTR),Y
0202   0756  C8               INY
0203   0757  CA               DEX
0204   0758  D0 F8            BNE   INNER
0205   075A  E6 16            INC   TABLEN   ; INCREMENTA LA LUNGHEZZA
                                             ; TABELLA.
0206   075C  A2 FF            LDX   # $ FF
```

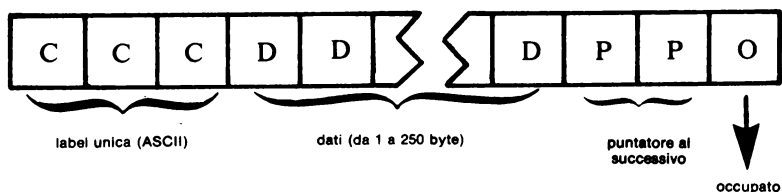
Figura 9-22. Programma della Lista Alfabetica: Ricerca Binaria, Cancellazione, Inserzione (continua)

```
0207  075E 60          OUTE    RTS               ; Z = 1 SE NON FATTO.
0208  075F             ;
0209  075F             ;
0210  075F             ;
0211  075F 20  00  06  DELETE  JSR  SEARCH       ; ACCETTA ADDR
                                                 ; DELL'OGGETTO NELLA
                                                 ; TABELLA.
0212  0762 D0  35              BNE  OUTS         ; VEDI SE È LA'
0213  0764 A5  16              LDA  TABLEN       ; VEDI QUANTI INGRESSI
0214  0766 38                  SEC               ; ...SONO DOPO L'OGGETTO
                                                 ; NELLA TABELLA.
0215  0767 E5  17              SBC  LOGPOS
0216  0769 F0  2A              BEQ  DECER
0217  076B 85  17              STA  LOGPOS       ; MEMORIZZA IL RISULTATO
                                                 ; COME CONTATORE.
0218  076D A5  18      BIGLOP  LDA  ENTLEN       ; PONI TEMP ED ENTRY UN
                                                 ; INGRESSO SOPRA
                                                 ; L'OGGETTO.
0219  076F 18                  CLC
0220  0770 65  14              ADC  POINTR
0221  0772 85  19              STA  TEMP
0222  0774 A9  00              LDA  # 0
0223  0776 65  15              ADC  POINTR+1
0224  0778 85  1A              STA  TEMP+1
0225  077A A6  1B              LDX  ENTLEN       ; POSIZIONA I CONTATORI.
0226  077C A0  00              LDY  # 0
0227  077E B1  19      BYTE    LDA  (TEMP),Y     ; MUOVI UN BYTE.
0228  0780 91  14              STA  (POINTR),Y
0229  0782 C8                  INY               ; IL BLOCCO È ANCORA
                                                 ; MOSSO?
0230  0783 CA                  DEX
0231  0784 D0  F8              BNE  BYTE
0232  0786 A5  18              LDA  ENTLEN
0233  0788 18                  CLC
0234  0789 65  14              ADC  POINTR
0235  078B 85  14              STA  POINTR
0236  078D 90  02              BCC  D2
0237  078F E6  15              INC  POINTR+1
0238  0791 C6  17      D2      DEC  LOGPOS
0239  0793 D0  D8              BNE  BIGLOP
0240  0795 C6  16      DECER   DEC  TABLEN
0241  0797 A9  00              LDA  # 0          ; Z = 1 SE FATTO.
0242  0799 60          OUTS    RTS
0243  079A                     .END
ERRORS = 0000 < 0000 >
SYMBOL TABLE
SYMBOL  VALUE

AD1     0688  ADDER   067B  ANOTHR  072C  BIGLOP  076D
BYTE    077E  CLOSE   0010  CMPRES  0011  D2      0791
DECER   0795  DELETE  075F  DIV     0615  ENTLEN  001B
ENTRY   062F  FOUND   06E2  INCMNT  0018  INNER   0752
INSERT  074E  LOGPOS  0017  LOOP    0621  LOOP    062C
LOSIDE  06F5  M1      0740  MAKCLO  0668  MOVER   072A
NEW     06E3  NEXT    066C  NOGOOD  064F  OBJECT  001C
OUT     06E0  OUTE    075E  OUTS    0799  POINTR  0014
SEARCH  0600  SET0    0717  SET1    0724  SETCLO  06D5
SETEMP  071A  SETUP   0700  SUB0    06BD  SUBIT   06A5
SUBLOP  06B2  TABASE  0012  TABLEN  0016  TEMP    0019
TESTS   0655  TOOHI   0695  TOOLOW  06C3  UPLOOP  070C

END OF ASSEMBLY
```

Figura 9-22. Programmi della Lista Alfabetica: Ricerca Binaria, Cancellazione, Inserzione.

Esaminiamo più dettagliatamente la struttura di Fig. 9-23. Il formato dell'ingresso è:

Come al solito le convenzioni sono:

ENTLEN: lunghezza totale dell'elemento (in byte)
TABASE: indirizzo della base della lista
TABLEN: numero di ingressi (da 1 a 256)

Si assume sempre che l'indirizzo di OBJECT risieda nel registro Y, prima dell'ingresso del programma.

In questo caso REFBASE punta all'indirizzo della base del direttorio, o "tabella di riferimento".

Ogni indirizzo a due byte all'interno del direttorio punta alla lettera corrispondente della lista. Quindi ogni gruppo di ingressi aventi la prima lettera uguale nella label, formano, in effetti, una lista separata all'interno dell'intera struttura. Questo facilita la ricerca ed è analogo ad un elenco indirizzi. Si noti che nessun dato viene mosso durante un'inserzione od una cancellazione. Solo i puntatori vengono variati, come avviene in ogni linked list ben realizzata.

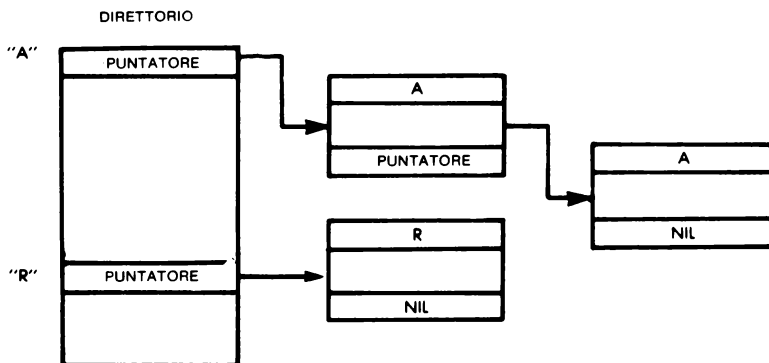



Figura 9.23: Struttura Linked List

Se non esiste nessun ingresso in corrispondenza ad una lettera particolare oppure se non esistono ingressi alfabetici a partire da un certo punto, i puntatori delle lettere corrispondenti punteranno all'inizio della tabella (="NIL"). Per convenzione, in fondo alla tabella viene memorizzato un valore tale che il valore assoluto della differenza tra quest'ultimo e "Z" sia maggiore della differenza tra "A" e "Z". Questo rappresenta il contrassegno di fine tabella (EOT: End Of Table). Qui si assume che il valore EOT occupi la stessa quantità di memoria di un ingresso normale ma potrebbe essere proprio un byte, se richiesto.

Si assume inoltre che le lettere alfabetiche siano codificate in ASCII. In caso contrario occorre variare la costante nella routine PRETAB.

Il contrassegno di fine tabella EOT è posto uguale al valore dell'inizio della tabella ("NIL").

Per convenzione i "puntatori NIL" che si trovano alla fine di una stringa o all'interno di una locazione di direttorio, e che non puntano ad una stringa, vengono posti uguali al valore della base della tabella per fornire un'identificazione unica. Si potrebbe utilizzare una convenzione alternativa. In particolare, un diverso contrassegno per EOT potrebbe far risparmiare dello spazio, se non è necessario nessun ingresso NIL per ingressi non esistenti.

L'inserzione e la cancellazione vengono eseguite nel modo consueto (vedere la parte I di questo capitolo) mediante la modifica diretta dei puntatori richiesti. Il flag INDEXD viene utilizzato per indicare se il puntatore all'oggetto si trova nella tabella di riferimento oppure in un altro elemento della stringa.

### Ricerca

Il programma di ricerca SEARCH risiede nelle locazioni di memoria da 0600 a 0650. Inoltre esso utilizza la subroutine PRETAB che si trova all'indirizzo 06F8.

Il principio di ricerca è immediato:

1 - accetta l'ingresso del direttorio corrispondente alla lettera dell'alfabeto nella prima posizione della label di OBJECT.

2 - accetta il puntatore di uscita del direttorio. Accetta l'elemento. Se NIL l'ingresso non esiste.

3 - se non NIL, si confronta l'elemento con OBJECT. Se sono uguali, la ricerca ha dato risultato positivo. Se sono diversi si accetta il puntatore all'ingresso successivo nella lista.

4 - ritorna al passo 2.

La Fig. 9-24 mostra un esempio di questo algoritmo.

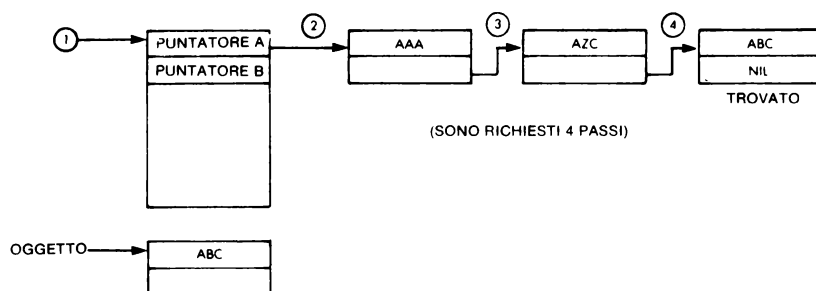



Figura 9.24: Linked List: una ricerca

### Inserzione di Elemento

L'inserzione è semplicemente una ricerca seguita da un'inserzione se si trova un "NIL". Un blocco di memoria per il nuovo ingresso viene allocato dopo il contrassegno EOT, purchè sia disponibile un contrassegno di posizione. Il programma si chiama "NEW" e risiede agli indirizzi da 0651 a 06BD. La Fig. 9-25 riporta un esempio.

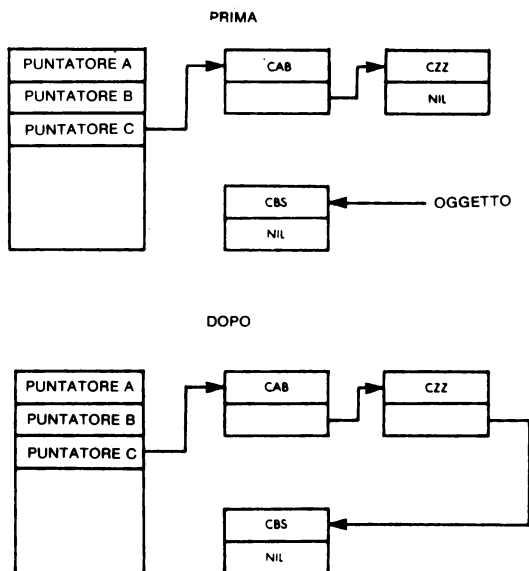



Figura 9.25: Linked List: esempio di inserzione

## Cancellazione di Elemento

Un elemento viene cancellato ponendo il suo contrassegno di posizione a "disponibile" e regolando il puntatore del testo dal direttorio o dall'elemento precedente. Il programma si chiama "DELETE" e risiede agli indirizzi da 06BE a 06F7. La Fig. 9-26 riporta un esempio di cancellazione.

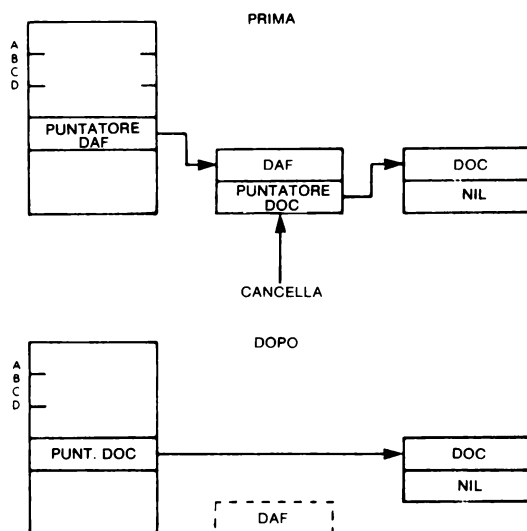



Figura 9.26: Esempio di cancellazione (Linked List)

```
LINEA  #LOC  CODICE      LINEA

0002   0000              INDEXB   = $10
0003   0000              INBLOC   = $11
0004   0000              POINTR   = $13
0005   0000              OBJECT   = $15
0006   0000              TEMP     = $17
0007   0000              REFBAS   = $19
0008   0000              OLD      = $1B
0009   0000              TABASE   = $1B
0010   0000              ENTLEN   = $1F
0011   0000              ;
0012   0000                       *=     $ 600
0013   0600              ;
0014   0600  A9 01       SEARCH   LDA     1            ; INIZIALIZZA I FLAG.
0015   0602  85 10                STA    INDEXB
0016   0604  20 F8 06             JSR    PRETAB        ; ACCETTA COME INIZIO REF.
                                                       ; PUNTATORE.
0017   0607  B1 11                LDA    (INDLOC),Y    ; METTILO IN POINTR.
0018   0609  85 13                STA    POINTR.
0019   060B  C8                   INY
0020   060C  B1 11                LDA    (INDLOC),Y
0021   060E  85 14                STA    POINTR+1
0022   0610  A0 00       ENTRY    LDY    # 0           ; VEDI SE L'INGRESSO È IL
                                                       ; VALORE EOT.
0023   0612  B1 13                LDA    (POINTR),Y
0024   0614  C9 7C                CMP     # $7C
0025   0616  F0 36                BEQ    NOTFND
0026   0618  B1 15                LDA    (OBJECT),Y    ; CONFRONTA LE PRIME
                                                       ; LETTERE.
0027   061A  D1 13                CMP    (POINTR), Y
0028   061C  90 30                BCC    NOTFND
0029   061E  D0 12                BNE    NOGOOD
0030   0620  C8                   INY                  ; CONFRONTA LE SECONDE
                                                       ; LETTERE.
0031   0621  B1 15                LDA    (OBJECT),Y
0032   0623  D1 13                CMP    (POINTR),Y
0033   0625  90 27                BCC    NOTFND
0034   0627  D0 09                BNE    NOGOOD
0035   0629  C8                   INY                  ; CONFRONTA LE TERZE
                                                       ; LETTERE.
0036   062A  B1 15                LDA    (OBJECT),Y
0037   062C  D1 13                CMP    (POINTR),Y
0038   062E  90 1E                BCC    NOTFND
0039   0630  F0 1E                BEQ    FOUND
0040   0632  A5 14       NOGOOD   LDA    POINTR+1      ; CONSERVA POINTR COME
                                                       ; RIFERIMENTO.
0041   0634  85 1C                STA    OLD+1
0042   0636  A5 13                LDA    POINTR
0043   0638  85 1B                STA    OLD
0044   063A  A4 1F                LDY    ENTLEN        ; ACCETTA PUNTATORE DA
                                                       ; INGRESSO E
0045   063C  B1 13                LDA    (POINTR),Y    ... CARICALO IN POINTR.
0046   063E  AA                   TAX
0047   063F  C8                   INY
0048   0640  B1 13                LDA    (POINTR),Y
0049   0642  85 14                STA    POINTR+1
0050   0644  8A                   TXA
0051   0645  85 13                STA    POINTR
0052   0647  A9 00                LDA     0
0053   0649  85 10                STA    INDEXD        ; RESET FLAG.
0054   064B  4C 10 06             JMP    ENTRY
```

Figura 9.27: Programma Linked List (continua).

```
0055   064E   A9 FF      NOTFND  LDA   #$FF
0056   0650   60         FOUND   RTS              ; Z = 1 SE TROVATO
0057   0651                      ;
0058   0651                      ;
0059   0651                      ;
0060   0651   20 00 06   NEW     JSR   SEARCH     ; VEDI SE L'OGGETTO È GIA'
                                                  ; LA'
0061   0654   F0 67              BEQ   OUTE
0062   0656   A5 1D              LDA   TABASE     ; CERCA UN ... BLOCCO
0063   0658   18                 CLC              ; INGRESSO NON OCCUPATO.
0064   0659   69 01              ADC   #1         ; SALTA DOPO IL VALORE EOT.
0065   065B   85 17              STA   TEMP
0066   065D   A9 00              LDA   # 1
0067   065F   65 1E              ADC   TABASE+1
0068   0661   85 18              STA   TEMP+1
0069   0663   A4 1F              LDY   ENTLEN     ; POSIZIONA Y PER PUNTARE
                                                  ; AL
0070   0665   C8                 INY              ; ... MARKER DI OCCUPAZIONE
                                                  ; DI UN INGRESSO.
0071   0666   C8                 INY
0072   0667   A9 01      LOOPP   LDA    1         ; TEST PER IL MARKER DI
                                                  ; OCCUPAZIONE.
0073   0669   D1 17              CMP   (TEMP),Y
0074   066B   D0 16              BNE   INSERT
0075   066D   A5 17              LDA   TEMP       ; SE UTILIZZATO, MUOVI
                                                  ; TEMP AL SUCCESSIVO.
0076   066F   18                 CLC              ; ... BLOCCO D'INGRESSO.
0077   0670   65 1F              ADC   ENTLEN
0078   0672   90 02              BCC   MORE
0079   0674   E6 18              INC   TEMP+1
0080   0676   69 03      MORE    ADC    3
0081   0678   85 17              STA   TEMP
0082   067A   A9 00              LDA   #0
0083   067C   65 18              ADC   TEMP+1
0084   067E   85 18              STA   TEMP+1
0085   0680   4C 67 06           JMP   LOOP
0086   0683   88         INSERT  DEY              ; PONI Y INDIETRO PER
                                                  ; PUNTARE
0087   0684   88                 DEY              ; ... ALLA SOMMITA' DEI DATI.
0088   0685   88         LOPE    DEY              ; TRASFERISCI L'OGGETTO
                                                  ; NELLO SPAZIO.
0089   0686   N1 15              LDA   (OBJECT),Y
0090   0688   91 17              STA   (TEMP),Y
0091   068A   C0 00              CPY    0
0092   068C   D0 F7              BNE   LOPE
0093   068E   A4 1F              LDY   ENTLEN     ; METTI IL VALORE DI POINTR,
0094   0690   A5 13              LDA   POINTR     ; L'OGGETTO DOPO
                                                  ; L'INGRESSO,
0095   0692   91 17              STA   (TEMP),Y   ; NELL'AREA PUNTATORE
                                                  ; DELL'OGGETTO.
0096   0694   C8                 INY
0097   0695   A5 14              LDA   POINTR+1
0098   0697   91 17              STA   (TEMP),Y
0099   0699   C8                 INY
0100   069A   A9 01              LDA   # 1        ; PONI AD 1 IL MARKER DI
                                                  ; OCCUPAZIONE.
0101   069C   91 17              STA   (TEMP),Y
0102   069E   A5 10              LDA   INDEXD     ; TEST PER VEDERE SE
                                                  ; TABELLA REF.
0103   06A0   D0 0D              BNE   SETINX     ; NECESSITA DI
                                                  ; RIAGGIUSTAMENTO.
0104   06A2   88                 DEY
0105   06A3   A5 18              LDA   TEMP+1     ; NO, CAMBIA IL PUNTATORE
```

**Figura 9.27: Programma Linked List (continua)**

```
0106   06A5  91  1B              STA   (OLD),Y      ; .. DELL'INGRESSO
                                                    ; PRECEDENTE.
0107   06A7  88                  DEY
0108   06A8  A5  17              LDA   TEMP
0109   06AA  91  1B              STA   (OLD),Y
0110   06AC  4C  88  06          JMP   DONE
0111   06AF  20  F8  06  SETINX  JSR   PRETAB       ; ACCETTA L'INDIRIZZO DI
                                                    ; CIO' CHE DEVE ESSERE
                                                    ; CAMBIATO.
0112   06B2  A5  17              LDA   TEMP         ; CARICA QUI L'INDIRIZZO
                                                    ; DELL'OGGETTO.
0113   06B4  91  11              STA   (INDLOC),Y
0114   06B6  C8                  INY
0115   06B7  A5  18              LDA   TEMP+1
0116   06B9  91  11              STA   (INDLOC),Y
0117   06BB  A9  FF      DONE    LDA   #$FF
0118   06BD  60          OUTE    RTS                ; Z = 0 SE FATTO
0119   06BE              ;
0120   06BE              ;
0121   06BE              ;
0122   06BE  20  00  06  DELETE  JSR   SEARCH       ; ACCETTA L'INDIRIZZO
                                                    ; DELL'OGGETTO.
0123   06C1  B0  34              BNE   OUTS
0124   06C3  A4  1F              LDY   ENTLEN       ; MEMORIZZA IL PUNTATORE
                                                    ; ALLA FINE
0125   06C5  B1  13              LDA   (POINTR),Y   ; ... DELL'OGGETTO.
0126   06C7  85  17              STA   TEMP
0127   06C9  C8                  INY
0128   06CA  B1  13              LDA   (POINTR),Y
0129   06CC  85  18              STA   TEMP+1
0130   06CE  C8                  INY
0131   06CF  A9  00              LDA   # 0          ; AZZERA IL MARKER DI
                                                    ; OCCUPAZIONE.
0132   06D1  91  13              STA   (POINTR),Y
0133   06D3  A5  10              LDA   INDEXD       ; VEDI SE TABELLA REF
                                                    ; NECESSITA
0134   06B5  F0  06              BEQ   PREINX       ; ... DI RIAGGIUSTAMENTO.
0135   06D7  20  F8  06          JSR   PRETAB
0136   06DA  4C  EA  06          JMP   MOVEIT
0137   06DD  A5  1B      PREINX  LDA   OLD          ; PREDISPONITI PER
                                                    ; CAMBIARE
0138   06DF  18                  CLC                ; ... L'INGRESSO PRECEDENTE.
0139   06E0  65  IF              ADC   ENTLEN
0140   06E2  85  11              STA   INDLOC
0141   06E4  A9  00              LDA   # 0
0142   06E6  65  IC              ADC   OLD+1
0143   06E8  85  12              STA   INDLOC+1
0144   06EA  A5  17      MOVEIT  LDA   TEMP         ; CAMBIA CIO' CHE
                                                    ; VA CAMBIATO.
0145   06EC  A0  00              LDY   # 0
0146   06EE  91  11              STA   (INDLOC),Y
0147   06F0  C8                  INY
0148   06F1  A5  18              LDA   TEMP+1
0149   06F3  91  11              STA   (INDLOC),Y
0150   06F5  A9  00              LDA   # 0
0151   06F7  60          OUTS    RTS                ; Z = 1 SE FATTO.
0152   06F8              ;
0153   06F8              ;
0154   06F8              ;
0155   06F8  A0  00      PRETAB  LDY   # 0
0156   06FA  B1  15              LDA   (OBJECT),Y
```

Fig. 9.27: Programma Linked List (continua).

```
0157   06FC  38          SEC              ; TOGLI LA TESTATA
                                          ; ASCII DALLA
0158   06FD  E9  41      SBC   #$41       ; ... PRIMA LETTERA
                                          ; DELL'OGGETTO.
0159   06FF  0A          ASL   A          ; MOLTIPLICA PER 2.
0160   0700  18          CLC
0161   0701  65  19      ADC   REFBAS     ; INDICE NELLA
                                          ; TABELLA REF.
0162   0703  85  11      STA   INDLOC     ;
0163   0705  A9  00      LDA   #0
0164   0707  65  1A      ADC   REFBAS+1
0165   0709  85  12      STA   INDLOC+1
0166   070B  60          RTS
0167   070C              .END

ERRORS = 0000 < 0000 >


SYMBOL TABLE

SYMBOL  VALUE

DELETE  06BE  DONE    06BB  ENTLEN  001F  ENTRY   061D
FOUND   0650  INDEXD  0010  INDLOC  0011  INSERT  0683
LOOP    0667  LOPE    0685  MORE    0676  MOVEIT  06EA
NEW     0651  NOGOOD  0632  NOTFND  064E  OBJECT  0015
OLD     001B  OUTE    06BD  OUTS    06F7  POINTR  0013
PREINX  06DD  PRETAB  06F8  REFBAS  0019  SEARCH  0600
SETINX  06AF  TABASE  001D  TEMP    0017
END OF ASSEMBLY
```

Figura 9.27: Programma Linked List.

## ALBERO BINARIO

Si svilupperanno ora delle routine tipiche di manipolazione ad albero. La Fig. 9-28 mostra una semplice struttura. I nomi verranno memorizzati internamente mediante etichette formate dalle prime tre lettere di ciascun nome. La rappresentazione di memoria di queste tre strutture appare in Fig. 9-29. Sono noti i contenuti dei nodi e dei due collegamenti di ciascuno di essi. Il primo collegamento, a sinistra del nome, è il "sibling di sinistra" ed il collegamento successivo, a destra, è il "sibling di destra". Per esempio, l'ingresso per Jones contiene due collegamenti: "2" e "4". Questo indica che il sibling di sinistra è l'ingresso numero 2 (Anderson) e quello di destra è il numero 4 (Smith). Uno zero, nel campo del collegamento, indica nessun sibling. L'etichetta del sibling di sinistra viene alfabeticamente prima di quello di destra.

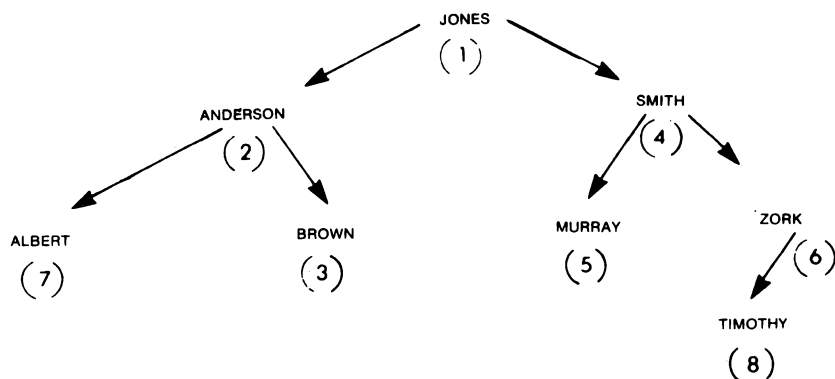



Figura 9.28: Albero binario

Le due routine principali per la manipolazione ad albero sono la *costruzione di albero* e l'*attraversamento di albero.* L'elemento da inserire verrà posizionato in un buffer. La routine di costruzione di albero inserirà i contenuti del buffer nell'albero in corrispondenza del nodo appropriato. La routine di attraversamento di albero, percorre recursivamente e stampa i contenuti di ogni nodo dell'albero, in ordine alfabetico. La Fig. 9-30 mostra il diagramma di flusso per la costruzione dell'albero e la Fig. 9-31 per l'attraversamento.

Poichè la routine per l'attraversamento è recursiva non si presta ad una rappresentazione mediante diagramma di flusso. Quindi si fornisce,

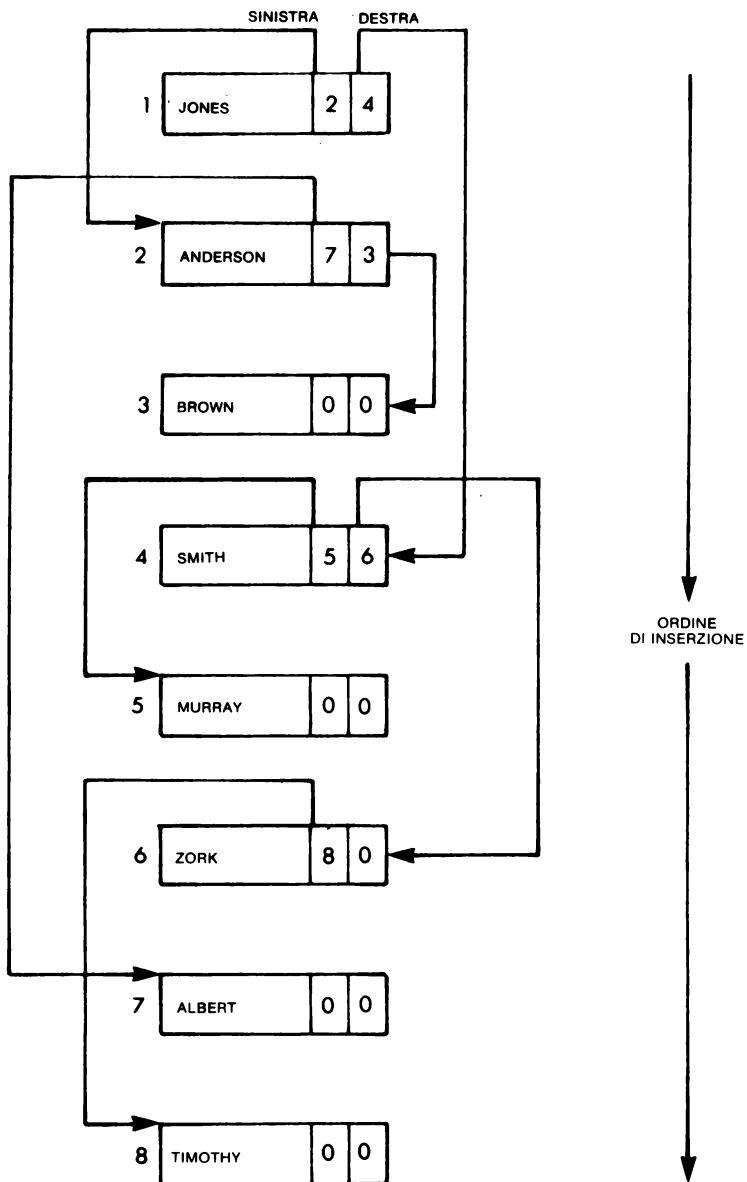



Figura 9.29: Rappresentazione in memoria

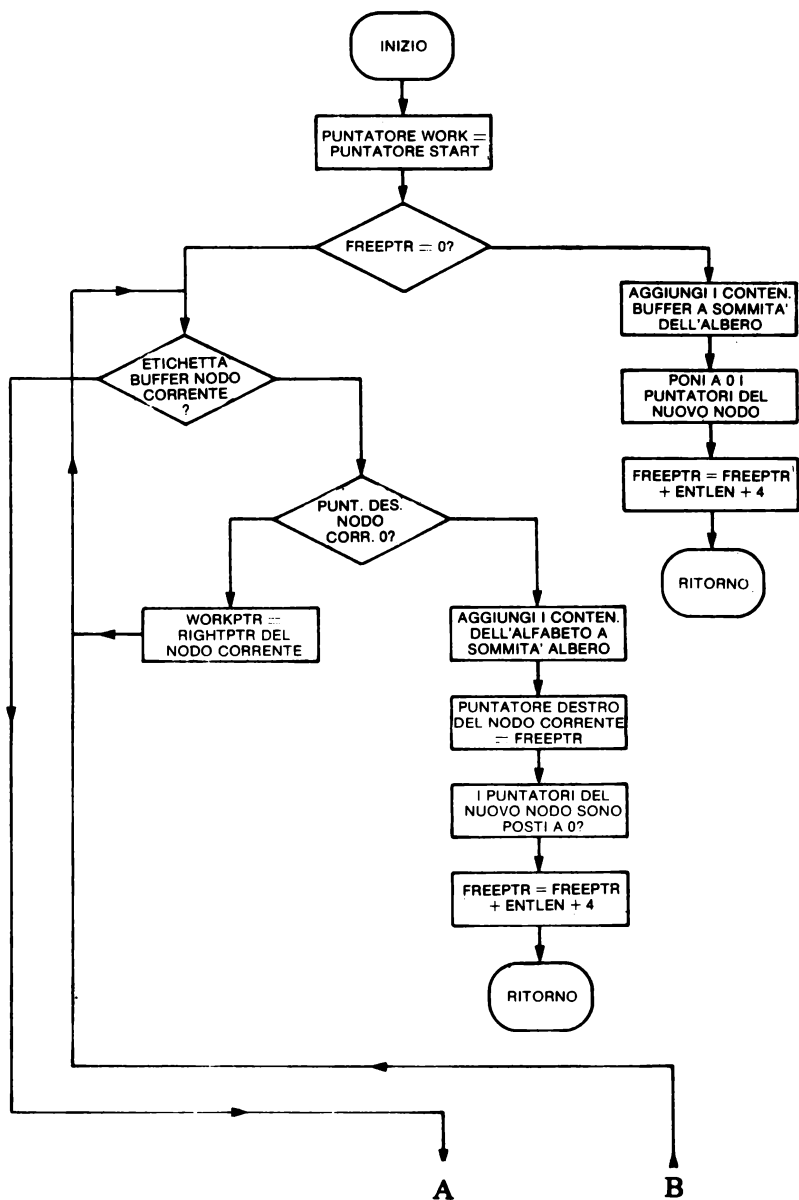



Figura 9.30: Diagramma di flusso della costruzione ad albero (continua).

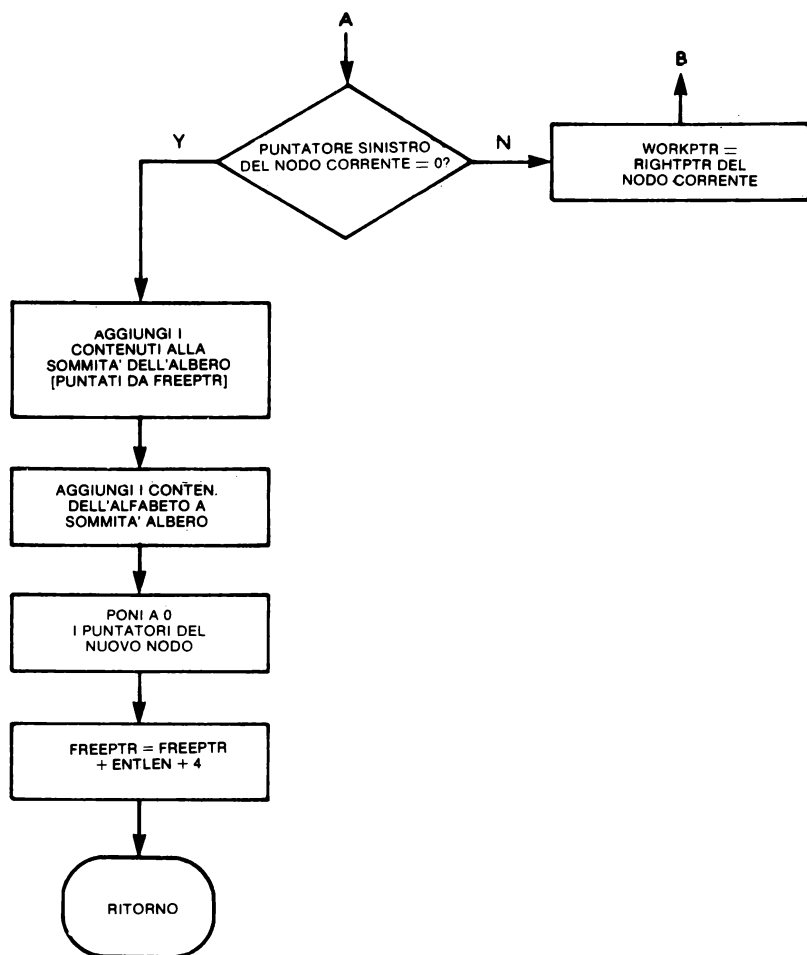



Figura 9.30: Diagramma di flusso della costruzione ad albero

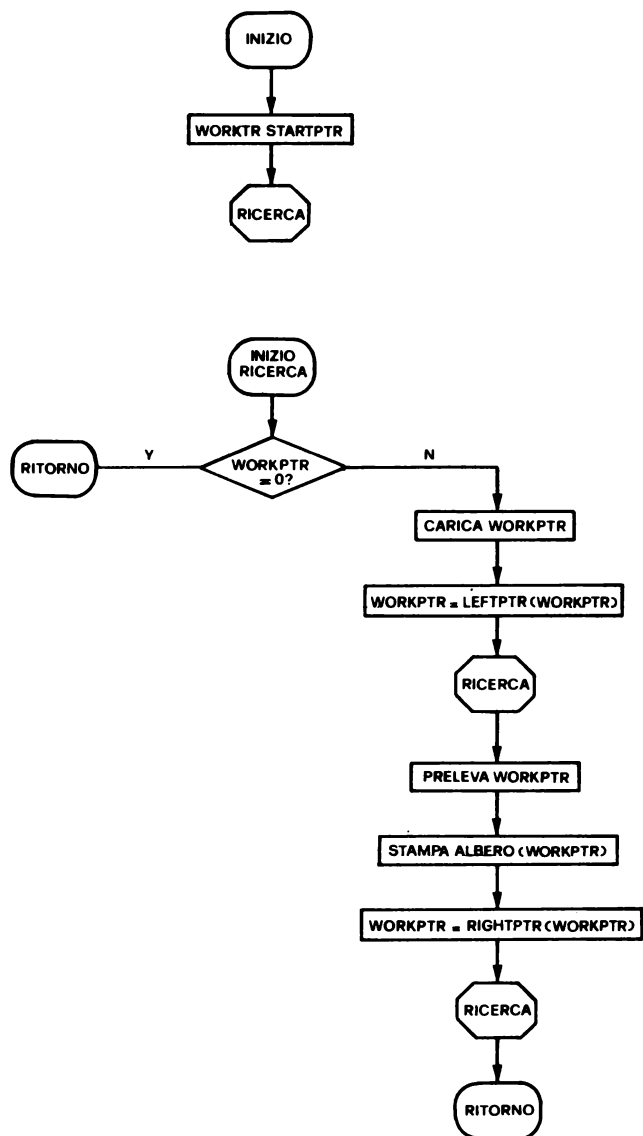



Figura 9.31: Diagramma di flusso dell'attraversamento di albero

```
PROGRAMMA PER ATTRAVERSAMENTO DI ALBERO
INIZIO
WORKPOINTER: = STARTPOINTER;
    RICERCA: INIZIO
             SE WORKPOINTER ≠0 ALLORA INIZIA
             INSERISCI WORKPTR;
             WORKPOINTER: = LEFTPTR (WORKPOINTER);
             PRELEVA WORKPOINTER;
             STAMPA [ALBERO (WORKPOINTER)];
             WORKPOINTER: = RIGHTPTR (WORKPOINTER);
             RICHIAMA RICERCA;
             FINE;
    RETURN;
    END;
END.
```

Figura 9.32: Algoritmo di attraversamento di albero

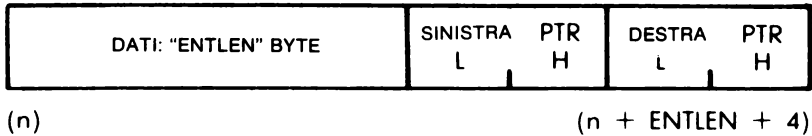



Figura 9.33: Unità dati o "Nodi" dell'albero

nella Fig. 9-32, un'altra rappresentazione della routine in un formato ad alto livello. La Fig. 9-33 mostra un nodo reale dell'albero. Essa contiene dei dati di lunghezza ENTLEN e due puntatori a 16 bit (il puntatore di destra e quello di sinistra). Per evitare confusione si noti che la rappresentazione della Fig. 9-29 è semplificata e che il puntatore di destra appare a sinistra nella memoria. La Fig. 9-34 mostra l'allocazione di memoria impiegata da questo programma e la Fig. 9-37 riporta il programma effettivo.

La routine INSERT risiede agli indirizzi da 02200 a 0282. L'etichetta dell'oggetto da inserire viene confrontata con l'ingresso. Se è maggiore ci si muove verso destra, se minore, a sinistra di una posizione. Il processo viene quindi ripetuto finchè non si trova un collegamento vuoto o si trova un "aggancio" adatto per un nuovo nodo (cioè un nodo è maggiore del successivo e viceversa). Il nuovo nodo viene quindi inserito con i collegamenti corretti.

La routine TRAVERSE risiede agli indirizzi da 0285 a 02D6. Le routine di servizio OUT, ADD e CLRPTR risiedono agli indirizzi da 0207 a 02FE (Vedere Fig. 9-37).

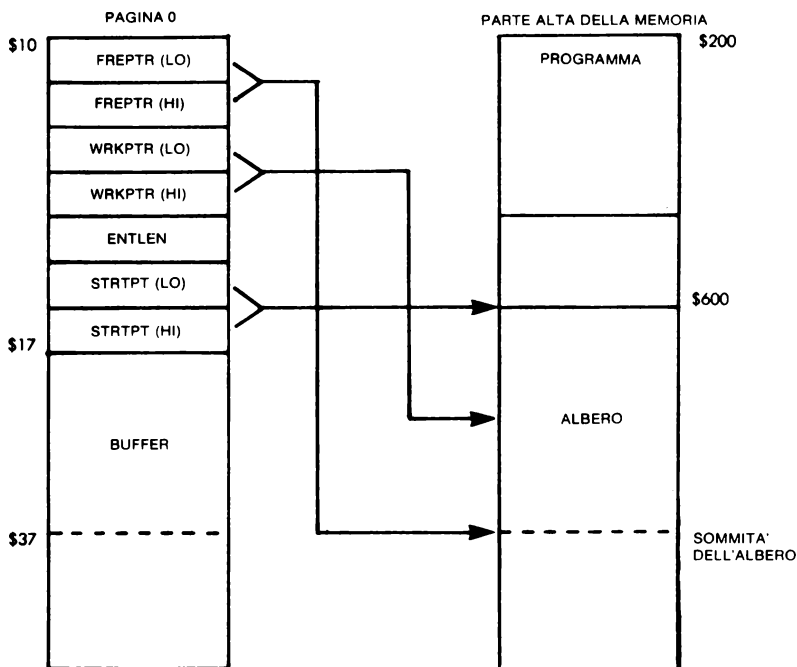



Figura 9.34: Mappe di memoria

La Fig. 9-35 mostra un esempio di inserzione di albero e la Fig. 9-36 mostra un esempio di attraversamento di albero.

## UN ALGORITMO HASHING

Un problema comune nella realizzazione di strutture dati è il posizionamento degli indicatori all'interno di un limitato spazio di memoria, in un modo schematico tale che possano essere facilmente recuperati. Sfortunatamente finchè gli indicatori sono dei numeri sequenziali distinti (senza vuoti) non si prestano al posizionamento in memoria.

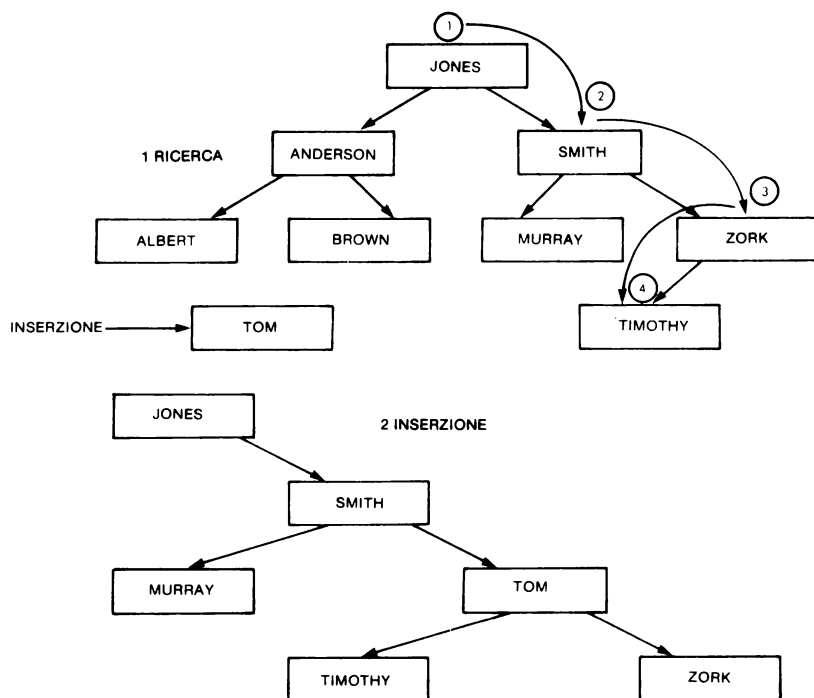



Figura 9.35: Inserzione di un elemento nell'albero

In particolare, se i nomi devono essere posizionati nella memoria così da poter essere recuperati più facilmente (cioè se essi sono posizionati alfabeticamente) essi potrebbero richiedere un'enorme quantità di memoria; per ogni nome possibile dovrebbe essere riservato un singolo blocco di memoria. La funzione matematica utilizzata per eseguire l'hashing dovrebbe essere semplice in modo da ottenere un algoritmo veloce, ma abbastanza sofisticato da rendere casuale la distribuzione dei nomi possibili sullo spazio di memoria disponibile. Il numero risultante può essere quindi utilizzato come un indice per la locazione effettiva e sarà possibile un recupero veloce. Questa è la ragione per cui l'hashing viene comunemente impiegato per le direttive dei nomi alfabetici.

Poichè nessun algoritmo può consentire di allocare due nomi nella stessa locazione di memoria (una "collisione") occorre escogitare una tecnica per risolvere il problema delle collisioni. Un buon algoritmo di

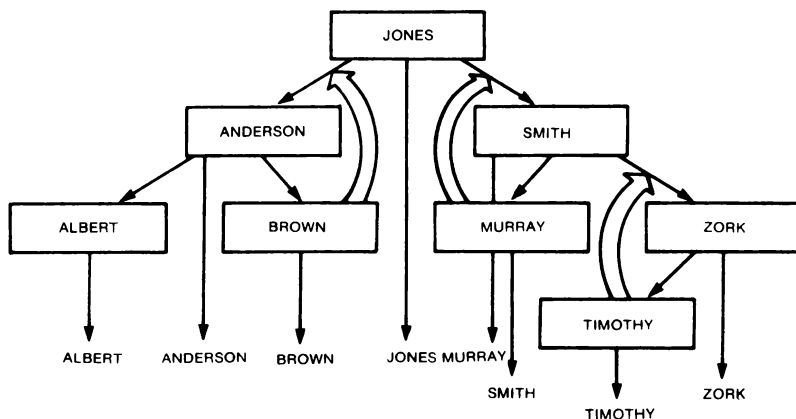



Figura 9.36: Listing dell'albero

hashing distribuirà i nomi eventualmente sullo spazio di memoria disponibile e consentirà una ricerca efficiente dei loro valori una volta che sono stati memorizzati in una tabella. L'algoritmo di hashing utilizzato in questa sede è molto semplice.

Esso esegue l'OR Esclusivo di tutti i byte della chiave. Per rendere ulteriormente casuale l'operazione viene eseguita una rotazione dopo ogni addizione.

La tecnica utilizzata per risolvere il problema delle collisioni è una semplice tecnica sequenziale. Essa tecnicamente viene chiamata "tecnica sequenziale di indirizzamento aperto"; il blocco disponibile sequenzialmente successivo viene allocato come ingresso. Questo può essere confrontato con un elenco di indirizzi tascabile. Si assume che si deve introdurre un nuovo ingresso come SMITH. Comunque nel nostro piccolo elenco di indirizzi la pagina "S" è piena. Si utilizzerà la pagina sequenzialmente successiva. (In questo caso la "T"). Si noti che necessariamente non ci sarà un'altra collisione con un nuovo ingresso iniziante con "T", l'ingresso "S" sarà rimosso prima che debba entrare una "T".

Inoltre si noti che potrebbe sussistere una catena di collisioni. Se la catena è lunga la tabella non è piena, l'algoritmo hashing è mal progettato.

```
LINEA  #LOC  CODICE     LINEA

0002   0000              ; PROGRAMMA DI MANAGEMENT DI ALBERO.
0003   0000              ; 2 ROUTINE: UNA, QUANDO CHIAMATA, PONE
0004   0000              ; NELL'ALBERO I CONTENUTI DEL
0005   0000              ; BUFFER; LA SECONDA ATTRAVERSA L'ALBERO
0006   0000              ; RECURSIVAMENTE, STAMPANDO, IN ORDINE
0007   0000              ; ALFABETICO I CONTENUTI DEI SUOI NODI.
0008   0000              ; NOTA: 'ENTLEN' VA INIZIALIZZATO
0009   0000              ; E 'FREPTR' DEVE ESSERE UGUALE A
0010   0000              ; 'STRTPTR' PRIMA DI IMPIEGARE ENTRAMBE LE ROUTINE.
0011   0000              ;
0012   0000                        *=$ 10
0013   0010              FREPTR    *=*+2            ; PUNTATORE SPAZIO LIBERO.
0014   0012                                ; PUNTA ALLA SUCCESSIVA LOCAZIONE DI
                                           ; MEMORIA LIBERA.
0015   0012              WRKPTR    *=*+2            ; WORK POINTER PUNTA AL
                                                    ; NODO CORRENTE.
0016   0014              ENTLEN    *=*+1            ; LUNGHEZZA INGRESSI
                                                    ; ALBERO, IN BYTE.
0017   0015  00 06       STRTPT    WORD $600
0018   0017              BUFFER    *=*+20           ; BUFFER DI I/O.
0019   002B              ;
0020   002B                        *=*$200
0021   0200              ;
0022   0200              ; ROUTINE DI COSTRUZIONE ALBERO: AGGIUNGI UN'UNITA'
                         ; DATI,
0023   0200              ; O NODO ALL'ALBERO. DEVE ESSERE
0024   0200              ; CHIAMATA CON L'UNITA DATI DA AGGIUNGERE IN 'BUFFER'.
0025   0200              ;
0026   0200  A5 15       INSERT    LDA  STRTPT      ; WORKPOINTER (=
                                                    ; FREEPOINTER).
0027   0202  85 12                 STA  WRKPTR
0028   0204  A5 16                 LDA  STRTPT +1
0029   0206  85 13                 STA  WRKPTR+1
0030   0208  A5 10                 LDA  FREPTR      ; SE FREEPOINTER
                                                    ; <>
0031   020A  C5 15                 CMP  STRTPT      ; PUNTATORE DELLA
                                                    ; LOCAZIONE DI PARTENZA.
0032   020C  D0 0D                 BNE  INLOOP      ; ENTRA NEL CICLO DI
                                                    ; INSERZIONE.
0033   020E  A5 11                 LDA  FREPTR+1
0034   0210  C5 16                 CMP  STRTPT+1
0035   0212  D0 07                 BNE  INLOOP
0036   0214  20 D7 02              JSR  ADD         ; CARICA BUFFER NELLA
                                                    ; POSIZIONE CORRENTE.
0037   0217  20 E4 02              JSR  CLRPTR      ; PONI A 0 I PUNTATORI DEL
                                                    ; NODO CORRENTE.
0038   021A  60                    RTS              ; FATTO AGGIUNGENDO IL
                                                    ; PRIMO NODO.
0039   021B  A0 00       INLOOP    LDY   #0         ; CONFRONTA L'ETICHETTA
                                                    ; DEL BUFFER A QUELLA
                                                    ; DELLA ...
0040   021D  B9 17 00    CMPLP     LDA  BUFFER,Y    ; LOCAZIONE CORRENTE.
0041   0220  D1 12                 CMP  (WRKPTR),Y
0042   0222  90 33                 BCC  LESSTN      ; ETICHETTA BFR PIU' BASSA:
0043   0224                                         ; AGGIUNGI BUFFER A PARTE
                                                    ; SINISTRA ALBERO.
0044   0224  F0 02                 BEQ  NXT         ; ETICHETTE UGUALI,
                                                    ; VERIFICA QUELLE DEL
                                                    ; CARATTERE SUCCESSIVO.
0045   0226  B0 05                 BCS  GRTNEQ      ; ETICHETTA BFR MAGGIORE;
                                                    ; AGGIUNGI BFR A
```

Figura 9-37. Programmi di Ricerca nell'Albero (continua).

```
0046  0228                                    ; PARTE DESTRA ALBERO.
0047  0228  C8          NXT     INY
0048  0229  C9 04               CMP   #4           ; 3 CARATTERI CONFRONTATI.
0049  022B  D0 F0               BNE   CMPLP        ; NO, CONTROLLA QUELLO
                                                   ; SUCCESSIVO.
0050  022D  A4 14       GRTNEQ  LDY   ENTLEN       ; IL PUNTATORE DESTRO
0051  022F  B1 12               LDA   (WRKPTR),Y   ; DEL NODO CORRENTE È = 0.
0052  0231  D0 15               BNE   NXRNOD       ; SE NO, MUOVITI
                                                   ; NELL'ALBERO IN BASSO
                                                   ; A DESTRA.
0053  0233  C8                  INY
0054  0234  B1 12               LDA   (WRKPTR),Y
0055  0236  D0 10               BNE   NXRNOD
0056  0238  A5 11               LDA   FREPTR+1     ; PONI IL PUNTATORE DESTRO
0057  023A  91 12               STA   (WRKPTR),Y   ; DEL NODO CORRENTE =
                                                   ; FREEPOINTER.
0058  023C  88                  DEY
0059  023D  A5 10               LDA   FREPTR
0060  023F  91 12               STA   (WRKPTR),Y
0061  0241  20 D7 02            JSR   ADD          ; AGGIUNGI BUFFER
                                                   ; ALL'ALBERO.
0062  0244  20 E4 02            JSR   CLRPTR       ; AZZERA I PUNTATORI DEL
                                                   ; NODO SUCCESSIVO.
0063  0247  60                  RTS                ; FATTO, AGGIUNTO NUOVO
                                                   ; NODO.
0064  0248  A4 14       NXRNOD  LDY   ENTLEN       ; POSIZIONA WORK POINTER.
0065  024A  B1 12               LDA   (WRKPTR),Y;  PUNTATORE A DESTRA DEL
                                                   ; NODO CORRENTE.
0066  024C  AA                  TAX
0067  024D  C8                  INY
0068  024E  B1 12               LDA   (WRKPTR),Y
0069  0250  85 13               STA   WRKPTR+1
0070  0252  86 12               STX   WRKPTR
0071  0254  4C 1B 02            JMP   INLOOP       ; PROVA IL NUOVO NODO
                                                   ; CORRENTE.
0072  0257  A4 14       LESSTN  LDY   ENTLEN       ; IL PUNTATORE DI SINISTRA
0073  0259  C8                  INY                ; DEL NODO CORRENTE È = 0?
0074  025A  C8                  INY
0075  025B  B1 12               LDA   (WRKPTR),Y
0076  025D  B0 15               BNE   NXLNOD       ; SE SI, MUOVITI IN BASSO A
                                                   ; SINISTRA NELL'ALBERO.
0077  025F  C8                  INY
0078  0260  B1 12               LDA   (WRKPTR),Y
0079  0262  B0 10               BNE   NXLNOD
0080  0264  A5 11               LDA   FREPTR+1     ; POSIZIONA IL PUNTATORE
                                                   ; DI SINISTRA DAL NODO
0081  0266  91 12               STA   (WRKPTR),Y   ; CORRENTE AL NUOVO NODO.
0082  0268  88                  DEY
0083  0269  A5 10               LDA   FREPTR
0084  026B  91 12               STA   (WRKPTR),Y
0085  026D  20 D7 02            JSR   ADD          ; AGGIUNGI I CONTENUTI DEL
                                                   ; NUOVO NODO.
0086  0270  20 E4 02            JSR   CLRPTR       ; AZZERA I PUNTATORI DEL
                                                   ; NUOVO NODO.
0087  0273  60                  RTS                ; FATTO, NUOVO NODO
                                                   ; AGGIUNTO.
0088  0274  A4 14       NXLNOD  LDY   ENTLEN       ; PONI WORKPOINTER =
                                                   ; PUNTATORE DI
0089  0276  C8                  INY                ; SINISTRA DEL NODO
                                                   ; CORRENTE.
0090  0277  C8                  INY
0091  0278  B1 12               LDA   (WRKPTR),Y
```

Figura 9-37. Programmi di Ricerca nell'Albero (continua).

```
0092   027A  AA                  TAX
0093   027B  C8                  INY
0094   027C  B1 12               LDA   (WRKPTR),Y
0095   027E  85 13               STA   WRKPTR+1
0096   0280  86 12               STX   WRKPTR
0097   0282  4C 1B02             JMP   INLOOP      ; PROVA IL NUOVO NODO
                                                   ; CORRENTE.
0098   0285             ;
0099   0285             ; ATTRAVERSAMENTO DI ALBERO: ELENCA I NODI
0100   0285             ; DELL'ALBERO IN ORDINE ALFABETICO.
0101   0285             ; È RICHIESTA LA ROUTINE D'USCITA PER
0102   0285             ; XFER BUFFER AL DISPOSITIVO D'USCITA.
0103   0285             ;
0104   0285  A5 15      TRVRSE   LDA   STRTPT      ; WORKING POINTER <=
                                                   ; START POINTER.
0105   0287  85 12               STA   WRKPTR
0106   0289  A5 16               LDA   STRTPT+1
0107   028B  85 13               STA   WRKPTR+1
0108   028D  A5 13      SEARCH   LDA   WRKPTR+1
0109   028F  A6 12               LDX   WRKPTR      ; SE WORK POINTER <> 0,
0110   0291  D0 07               BNE   OK          ; CONTINUA;
0111   0293  A4 13               LDY   WRKPTR+1
0112   0295  D0 03               BNE   OK
0113   0297  4C C6 02            JMP   RETN        ; ALTRIMENTI, RITORNO.
0114   029A  48         OK       PHA               ; SPINGI WORK POINTER
0115   029B  8A                  TXA               ; NELLO STACK.
0116   029C  48                  PHA
0117   029D  A4 14               LDY   ENTLEN      ; PONI WORKPONTER =
0118   029F  C8                  INY               ; PUNTATORE DI SINISTRA
                                                   ; DEL NODO CORRENTE.
0119   02A0  C8                  INY
0120   02A1  B1 12               LDA   (WRKPTR),Y
0121   02A3  AA                  TAX
0122   02A4  C8                  INY
0123   02A5  B1 12               LDA   (WRKPTR),Y
0124   02A7  85 13               STA   WRKPTR+1
0125   02A9  86 12               STX   WRKPTR
0126   02AB  20 8D 02            JSR   SEARCH      ;RICERCA RECURSIVAMENTE
                                                   ; IL NUOVO NODO.
0127   02AE  68                  PLA               ; PRELEVA IL VECCHIO NODO
                                                   ; CORRENTE E PONILO IN
                                                   ; WORK POINTER.
0128   02AF  85 12               STA   WRKPTR
0129   02B1  68                  PLA
0130   02B2  85 13               STA   WRKPTR+1
0131   02B4  20 C7 02            JSR   OUT         ; FA' USCIRE I CONTENUTI
                                                   ; DEL NODO CORRENTE.
0132   02B7  A4 14               LDY   ENTLEN      ; PONI WORK POINTER =
0133   02B9  B1 12               LDA   (WRKPTR),Y ; PUNTATORE DI DESTRA DEL
                                                   ; NODO CORRENTE.
0134   02BB  AA                  TAX
0135   02BC  C8                  INY
0136   02BD  B1 12               LDA   (WRKPTR),Y
0137   02BF  85 13               STA   WRKPTR+1
0138   02C1  86 12               STX   WRKPTR
0139   02C3  20 8D 02            JSR   SEARCH      ; RICERCA NUOVO NODO.
0140   02C6  60         RETN     RTS               ; FATTO, RITORNO.
0141   02C7             ;
0142   02C7             ; ROUTINE DI USCITA DEL BUFFER.
0143   02C7             ;
0144   02C7  A0 00      OUT      LDY    #0
0145   02C9  B1 12      XFR      LDA   (WRKPTR),Y ; ACCETTA CARATTERE DEL
                                                   ; NODO CORRENTE.
```

Figura 9-37. Programmi di Ricerca nell'Albero (continua).

```
0146  02CB  99 17 00           STA   BUFFER,Y    ; PONILO IN BUFFER.
0147  02CE  CB                 INY               ; RIPETI FINO ...
0148  02CF  C4 14              CPY   ENTLEN      ; AL TRASFERIMENTO DI
                                                 ; TUTTI I CARATTERI.
0149  02D1  D0 F6              BNE   XFR         ;
0150  02D3  EA                 NOP               ; INSERISCI LA CHIAMATA
                                                 ; ALLA
0151  02D4  EA                 NOP               ; SUBROUTINE CHE FA USCIRE
                                                 ; BUFFER.
0152  02D5  EA                 NOP
0153  02D6  60                 RTS               ; FATTO.
0154  02D7           ;
0155  02D7           ; ROUTINE CHE PONE I CONTENUTI
0156  02D7           ; DI BUFFER IN UN NUOVO NODO.
0157  02D7           ;
0158  02D7  A0 00    ADD       LDY   #0
0159  02D9  B9 17 00 NOV       LDA   BUFFER,Y    ; ACCETTA CARATTERE DA
                                                 ; BUFFER.
0160  02DC  91 10              STA   (FREPTR),Y  ; MEMORIZZALO IN UN NUOVO
                                                 ; NODO.
0161  02DE  C8                 INY               ; RIPETI FINO ...
0162  02DF  C4 14              CPY   ENTLEN      ; AL TRASFERIMENTO DI
                                                 ; TUTTI I CARATTERI.
0163  02E1  D0 F6              BNE   MOV
0164  02E3  60                 RTS               ; FATTO.
0165  02E4           ;
0166  02E4           ; ROUTINE PER AZZERARE I PUNTATORI DEL NUOVO NODO,
0167  02E4           ; E PER AGGIORNARE IL PUNTATORE DELLO SPAZIO LIBERO.
0168  02E4           ;
0169  02E4  A4 14    CLRPTR    LDY   ENTLEN      ; POSIZIONA L'INDICE PER
                                                 ; PUNTARE
0170  02E6                                       ; ALLA SOMMITA' DELLE
                                                 ; LOCAZIONI DEL PUNTATORE.
0171  02E6  A9 00              LDA   #0
0172  02E8  A2 04              LDX   #4          ; CICLA 4 VOLTE PER
                                                 ; AZZERARE I PUNTATORI.
0173  02EA  91 10    CLRLP     STA   (FREPTR),Y  ; AZZERA LA LOCAZIONE DEL
                                                 ; PUNTATORE.
0174  02EC  C8                 INY               ; PUNTA ALLA LOCAZIONE
                                                 ; SUCCESSIVA DEL
                                                 ; PUNTATORE.
0175  02ED  CA                 DEX
0176  02EE  D0 FA              BNE   CLRLP       ; RIRICLA SE NON FATTO.
0177  02F0  A5 14              LDA   ENTLEN      ; ACCETTA LA LUNGHEZZA
                                                 ; DELL'INGRESSO
0178  02F2  18                 CLC               ; ED AGGIUNGI 4 PER LO
                                                 ; SPAZIO PUNTATORE.
0179  02F3  69 04              ADC   #4
0180  02F5  65 10              ADC   FREPTR      ; AGGIUNGI AL PUNTATORE
0181  02F7  90 02              BCC   CC          ; DELLO SPAZIO LIBERO PER
                                                 ; AGGIORNARLO.
0182  02F9  E6 11              INC   FREPTR+1    ; ATTENZIONE AGLI
                                                 ; OVERFLOW
0183  02FB  85 10    CC        STA   FREPTR      ; RI-IMMAGAZZINA IL
                                                 ; PUNTATORE DELLO SPAZIO
                                                 ; LIBERO AGGIORNATO.
0184  02FD  60                 RTS               ; FATTO.
0185  02FE                     .END

ERRORS = 0000 < 0000 >
END OF ASSEMBLY
```

Figura 9-37. Programmi di Ricerca nell'Albero

## Nota sugli Alberi

Gli alberi binari possono essere costruiti ed attraversati in molti modi. Per esempio, una rappresentazione alternativa per l'albero che si considera potrebbe essere:

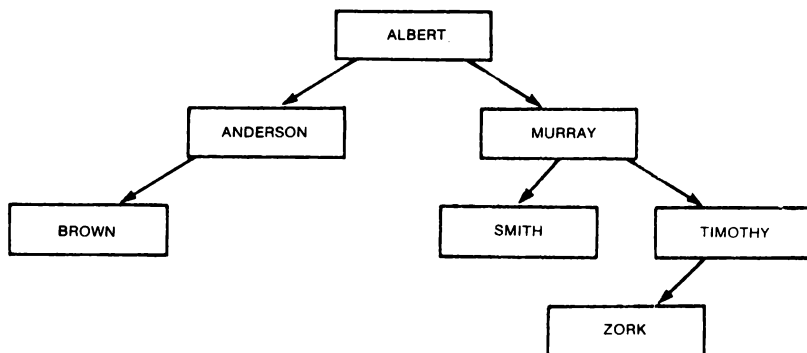



Figura 9.38: Albero in pre-ordine

Esso potrebbe essere attraversato in "preordine":

1 - elenca la radice
2 - attraversa il sub-albero di sinistra
3 - attraversa il sub-albero di destra

Esistono molte altre tecniche e convenzioni.

Poichè è conveniente utilizzare una potenza di due per il formato dei dati, la lunghezza dei dati è di otto caratteri; sei sono allocati come chiave e due come dati. Questa è una situazione tipica, per esempio, nella creazione della tabella dei simboli di un assembler. Fino a sei simboli esadecimali sono allocati come simbolo e due sono allocati per l'indirizzo che esso rappresenta (2 byte).

Nella ricerca di elementi da una tabella hashing, il tempo richiesto per la ricerca non dipende dalla dimensione della tabella ma dal grado di riempimento della stessa. Tipicamente, mantenendo la tabella piena meno dell'80%, si manterrà alto il tempo di accesso (uno o due tentativi). È responsabilità della routine chiamante mantenere la traccia del grado di riempimento della tabella e prevenire overflow.

L'aumento del tempo di accesso in funzione del riempimento della tabella è riportato in Fig. 9-39. Le routine principali utilizzate dal programma sono: quella di inizializzazione (INIT), mostrata in Fig. 9-40; la routine di memorizzazione, mostrata in Fig. 9-41; la routine di recupero, mostrata in Fig. 9-42 e la routine di hash, mostrata in Fig. 9-43. L'allocazione di memoria appare in Fig. 9-44 ed il programma in Fig. 9-45. Il programma ha lo scopo di mostrare tutti gli algoritmi principali utilizzati in un meccanismo di hashing reale. Se questi programmi fanno parte di una realizzazione effettiva si suggerisce vivamente di aggiungere le usuali funzioni richieste per prevenire situazioni particolari.

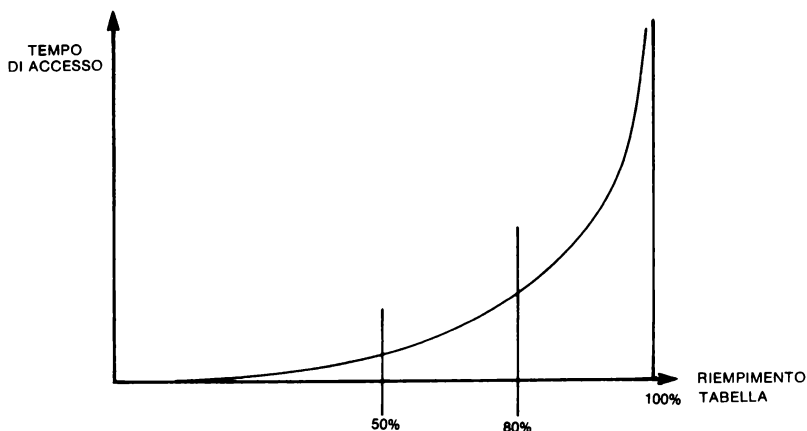



Figura 9.39: Tempo di accesso in funzione del riempimento relativo

In particolare, occorre salvaguardarsi contro l'evento di tabella piena o di una chiave non corretta poichè essi potrebbero causare dei cicli indefiniti nel programma. Si raccomanda vivamente al lettore di studiare questo programma. Infatti solo così si demistificherà l'algoritmo hashing ed inoltre si risolverà un problema pratico importante incontrato nel progetto di un assembler, o in qualsiasi struttura nella quale si debbano conservare delle tabelle di nomi con i loro equivalenti in un modo efficiente.

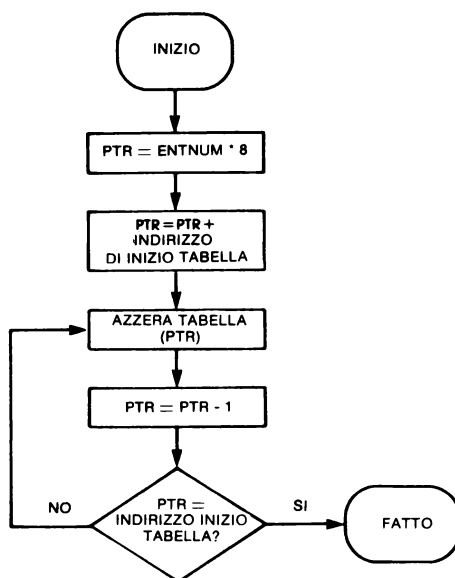



Figura 9.40: Subroutine di inizializzazione

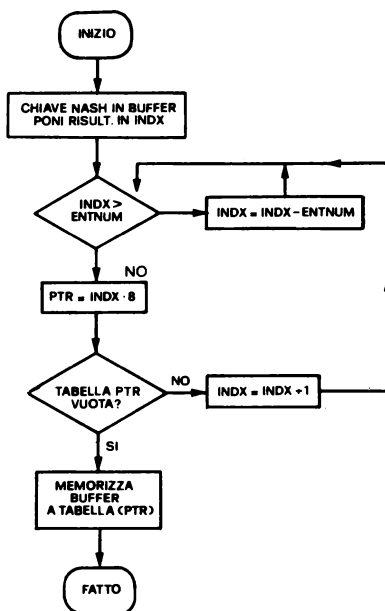



Figura 9.41: Routine di "memorizzazione"

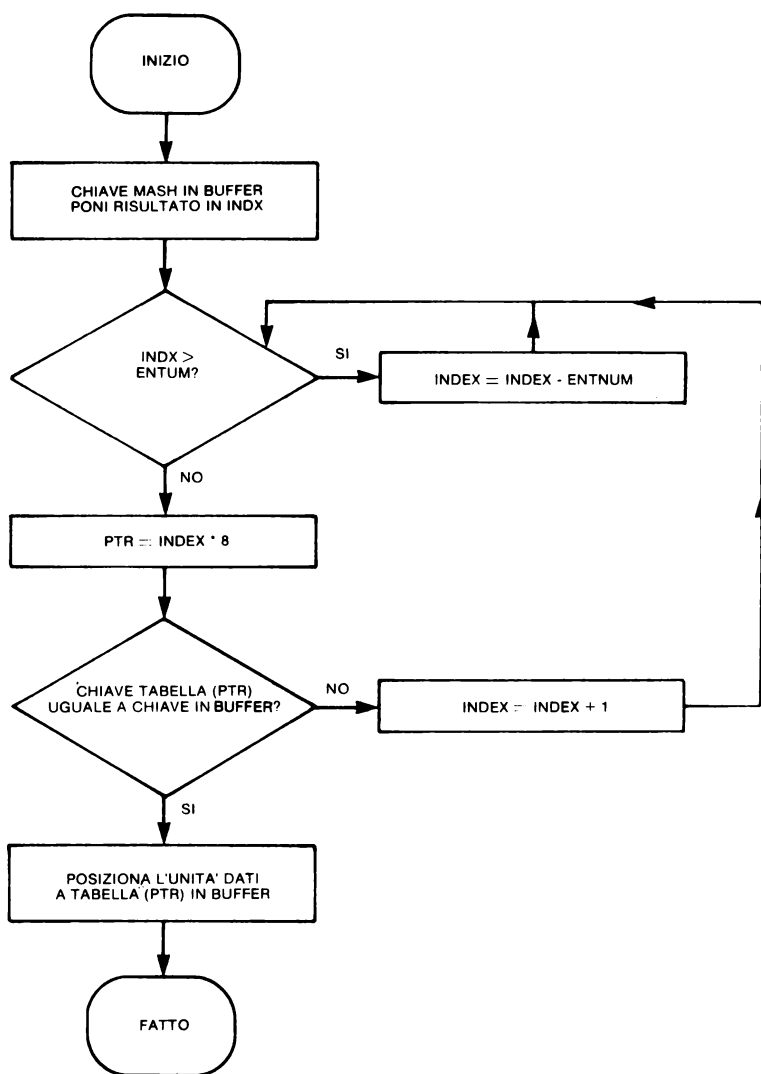



Figura 9.42: Routine di ricerca

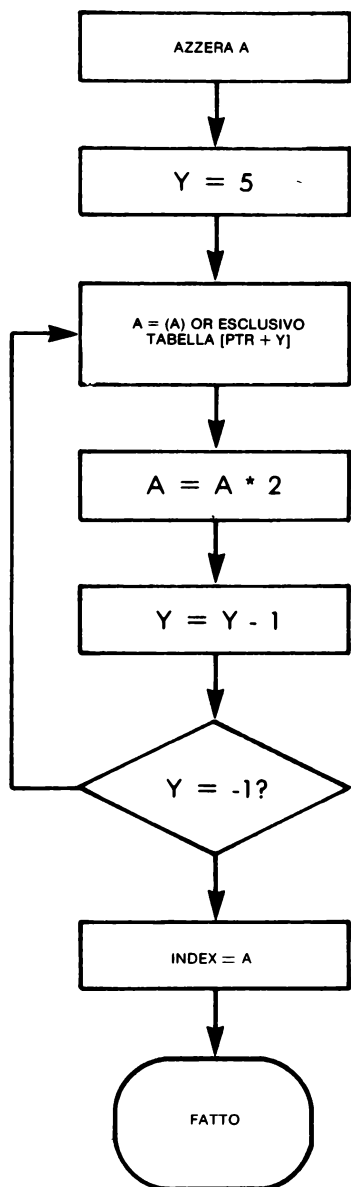



Figura 9.43: Hash routine

## BUBBLE-SORT

Bubble-sort è una tecnica di classificazione utilizzata per ordinare gli elementi in una tabella in ordine crecente o decrescente. La tecnica bubble-sort deriva il suo nome dal fatto che l'elemento più piccolo "bubble-up" (gorgoglia) alla sommità della tabella. Ogni volta che esso si scontra con un elemento "più pesante" esso lo scavalca.

La Fig. 9-46 mostra un esempio pratico di bubble-sort. La lista da classificare contiene: 10, 5, 0, 2 e 100 e deve essere ordinata in ordine crescente ("0" in alto). L'algoritmo è semplice ed il diagramma di flusso è mostrato in Fig. 9-47.

I due elementi alla sommità (o al fondo) vengono confrontati. Se l'elemento sottostante è il minore ("più leggero") allora questi vengono scambiati, in caso contrario no. In pratica lo scambio, se si verifica, sarà

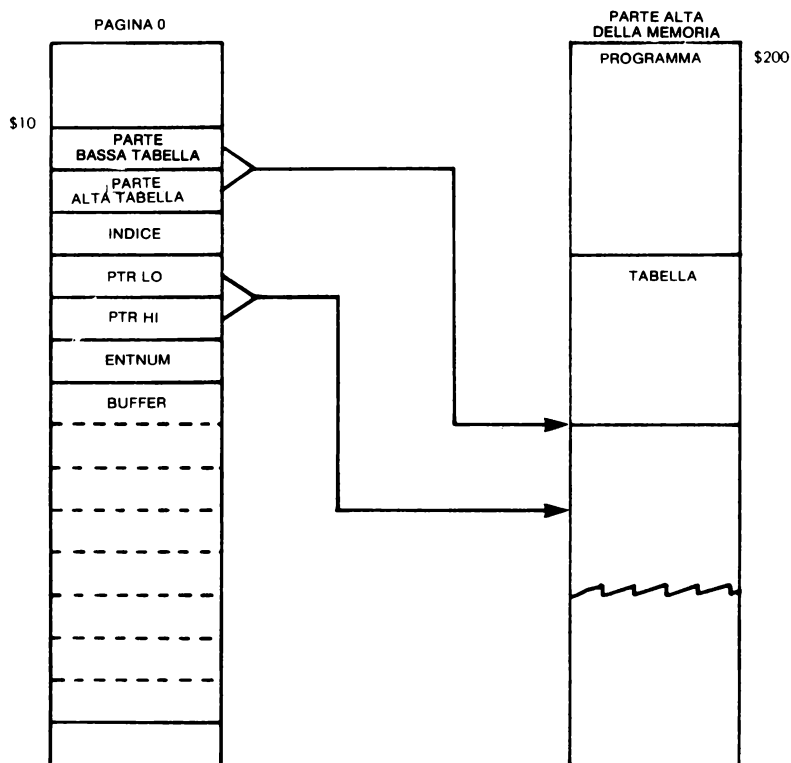



Figura 9.44: Mappe di memoria: memorizzazione/ricerca Hash

```
LINEA  #LOC  CODICE  LINEA

0002   0000          ; PROGRAMMA PER MEMORIZZARE I SIMBOLI
                     ; DELL'ASSEMBLATORE IN UNA
0003   0000          ; TABELLA, CUI SI ACCEDE MEDIANTE HASHING. I SIMBOLI
0004   0000          ; SONO 6 CAR, 2 DATI. IL NUMERO MASSIMO DI
0005   0000          ; UNITA' DI 8 BYTE DA MEMORIZZARE NELLA TABELLA
0006   0000          ; DOVREBBE ESSERE IN "ENTNUM" E L'INDIRIZZO
0007   0000          ; D'INIZIO DELLA TABELLA IN "TABLE". SI NOTI CHE
0008   0000          ; PRIMA DELL'IMPIEGO, OCCORRE INIZIALIZZARE
0009   0000          ; TABLE CON LA ROUTINE "INIT"
0010   0000          ; È RESPONSABILITA' DEL PROGRAMMA CHIAMANTE
0011   0000          ; NON SUPERARE LA DIMENSIONE TABELLA.
0012   0000          ;
0013   0000                    *=$ 10
0014   0010  00 06   TABLE     WORD $ 600     ; INDIRIZZO INIZIO TABELLA.
0015   0012          INDX      *=*+1          ; NUMERO UNITA', DATI
                                              ; DA ACCEDERE
0016   0013          PTR       *=*+2          ; PUNTATORE ALL'UNITA'
                                              ; DATI IN TABELLA.
0017   0015          ENTNUM    *=*+1          ; NUMERI DI INGRESSI IN
                                              ; TABELLA (256 MAX)
0018   0016          BUFFER    *=*+8          ; BUFFER D'INGRESSO/
                                              ; USCITA.
0019   001E
0020   001E                    *=$ 200
0021   0200          ;
0022   0200          ; ROUTINE "INIT": INIZIALIZZA A ZERO
0023   0200          ; LA TABELLA.
0024   0200          ;
0025   0200  A5 15   INIT      LDA  ENTNUM
0026   0202  85 13             STA  PTR       ; CARICA NUM. INGRESSI
                                              ; IN PUNTATORE
0027   0204  20 72 02          JSR  SHADD     ; MOLTIPLICA PTR*8,
                                              ; AGGIUNGI PUNTATORE
                                              ; TABELLA.
0028   0207  A2 00             LDX  #0        ; AZZERA X PER
                                              ; INDIRIZZAMENTO.
0029   0209  A9 00   CLRLP     LDA  #0        ; CARICA LA COSTANTE ZERO.
0030   020B  A4 13             LDY  PTR
0031   020D  D0 02             BNE  DECR      ; SE PTR < > 0, NON
                                              ; DECREMENTARE BYTE ALTO.
0032   020F  C6 14             DEC  PTR+1     ; DECREMENTA BYTE ALTO
                                              ; DEL PUNTATORE.
0033   0211  C6 13   DECR      DEC  PTR       ; DECREMENTA BYTE BASSO.
0034   0213  81 13             STA  (PTR,X)   ; AZZERA LOCAZIONE.
0035   0215  A5 13             LDA  PTR       ; CONTROLLA SE PUNTATORE
                                              ; = PUNTATORE TABELLA
0036   0217  C5 10             CMP  TABLE     ; SE DIVERSI, AZZERA
                                              ; LOCAZIONE SUCCESSIVA.
0037   0219  D0 EE             BNE  CLRLP
0038   021B  A5 14             LDA  PTR+1
0039   021D  C5 11             CMP  TABLE+1
0040   021F  D0 E8             BNE  CLRLP
0041   0221  60                RTS
0042   0222          ;
0043   0222          ; ROUTINE "STORE": POSIZIONA IN TABELLA I
0044   0222          ; CONTENUTI DI BUFFER USANDO IL PRIMO DI 6 CAR
0045   0222          ; DI BUFFER COME "CHIAVE" PER DETERMINARE
0046   0222          ; L'INDIRIZZO HASHED IN TABELLA.
0047   0222          ;
0048   0222  A2 00   STORE     LDX  # 0       ; AZZERA X PER
                                              ; INDIRIZZAMENTO
                                              ; INDICIZZATO.
```

Figura 9-45. Programma Hashing (continua)

```
0049   0224  20 90 02          JSR   HASH        ; ACCETTA INDICE HASHED
0050   0227  20 62 02  CMPR1   JSR   LIMIT       ; ASSICURATI CHE L'INDICE
                                                 ; È ENTRO I LIMITI.
0051   022A  A1 13             LDA   (PTR,X)     ; CONTROLLA L'UNITA' DATI ...
0052   022C  F0 05             BEQ   EMPTY       ; SALTA SE VUOTA.
0053   022E  E6 12             INC   INDX        ; PROVA L'UNITA' SUCCESSIVA.
0054   0230  4C 27 02          JMP   CMPR1       ; CONTROLLA SE L'INDICE
                                                 ; DELL'UNITA' SUCCESSIVA
                                                 ; È VALIDO.
0055   0233  A0 07     EMPTY   LDY   #7          ; RICICLA 8 VOLTE PER
                                                 ; CARICARE L'UNITA' DATI.
0056   0235  B9 16 00  FILL    LDA   BUFFER,Y    ; ACCETTA CAR. DAL BUFFER,
0057   0238  91 13             STA   (PTR), Y    ; POSIZIONALO NEL BUFFER.
0058   023A  88                DEY
0059   023B  10 F8             BPL   FILL        ; TRASFERISCI IL CAR.
                                                 ; SUCCESSIVO.
0060   023D  60                RTS               ; AGGIUNTA ESEGUITA.
0061   023E                    ;
0062   023E                    ; ROUTINE "FIND":
0063   023E                    ; CERCA L'INGRESSO LA CUI CHIAVE È NEL BUFFER.
0064   023E                    ; L'INGRESSO, SE TROVATO, È COPIATO NEL BUFFER.
0065   023E                    ; CON 2 BYTE DATI.
0066   023E                    ;
0067   067E  A2 00     FIND    LDX   #0          ; AZZERA X PER
                                                 ; INDIRIZZAMENTO INDIRETTO
0068   0240  20 90 02          JSR   HASH        ; ACCETTA IL PRODOTTO
                                                 ; HASH
0069   0243  20 62 02  CMPR2   JSR   LIMIT       ; ASSICURATI CHE IL
                                                 ; RISULTATO È ENTRO
                                                 ; I LIMITI.
0070   0246  A0 05             LDY   N5          ; RICICLA 6 VOLTE PER
                                                 ; CONFRONTARE BUFFER
                                                 ; CON DATI.
0071   0248  B1 13     CHLP    LDA   (PTR), Y    ; ACCETTA CARATTERE DA
                                                 ; TABELLA.
0072   024A  D9 16 00          CMP   BUFFER, Y   ; È = CAR. BUFFER?
0073   024D  D0 0E             BNE   BAD         ; SE NO, PROVA CON L'UNITA'
                                                 ; DATI SUCCESSIVA.
0074   024F  88                DEY
0075   0250  10 F6             BPL   CHKLP       ; CONTROLLA I CARATTERI
                                                 ; SUCCESSIVI.
0076   0252  A0 07     HATCH   LDY   N7          ; CICICLA 8 VOLTE PER
                                                 ; TRASFERIRE I CARATTERI
                                                 ; DEL BUFFER.
0077   0254  B1 13     XFER    LDA   (PTR), Y    ; ACCETTA CAR. DALLA
                                                 ; TABELLA.
0078   0256  99 16 00          STA   BUFFER, Y   ; MEMORIZZA NEL BUFFER.
0079   0259  88                DEY
0080   025A  10 FB             BPL   XFER        ; RIRICLA I CARATTERI
                                                 ; TRASFERITI.
0081   025C  60                RTS               ; FATTO: UNITA' DATI
                                                 ; TROVATA NEL BUFFER.
0082   025D  E6 12     BAD     INC   INDX        ; NON TROVATO, PROVA
                                                 ; L'UNITA' DATI SUCCESSIVA.
0083   025F  4C 43 02          JMP   CMPR2       ; NON VALIDA L'INDICE DELLA
                                                 ; NUOVA UNITA' DATI.
0084   0262                    ;
0085   0262                    ; ROUTINE PER ASSICURARSI CHE L'INDICE DATI
0086   0262                    ; È ENTRO I LIMITI DI ENTNUM. QUINDI
0087   0262                    ; MOLTIPLICA L'INDICE PER 8 E LO SOMMA AL PUNTATORE
0088   0262                    ; DELLA TABELLA. IL RISULTATO È POSIZIONATO IN "PTR"
                               ; COME INDIRIZZO UNITA' DATI.
0089   0262                    ;
```

Figura 9-45. Programma Hashing (continua)

```
0090   0262  A5 12     LIMIT   LDA  INDX        ; ACCETTA INDICE.
0091   0264  C5 15     TEST    CMP  ENTNUM      ; INDICE > NUMERO DATI?
0092   0266  90 06             BCC  OK          ; SALTA SE NO.
0093   026B  38                SEC              ; SI.
0094   0269  E5 15             SBC  ENTNUM      ; SOTTRAI N. DATI FINCHÈ
0095   026B  4C 64 02          JMP  TEST        ; INDICE SENZA LIMITI.
0096   026E  85 13     OK      STA  PTR         ; MEMORIZZA INDICE
                                                ; CORRETTO IN PUNTATORE.
0097   0270  85 12             STA  INDX        ; SALVA L'INDICE
                                                ; AGGIORNATO.
0098   0272  A9 00     SHADD   LDA  #0          ; AZZERA IL PUNTATORE
                                                ; SUPERIORE PER
                                                ; SCORRIMENTI.
0099   0274  85 14             STA  PTR+1
0100   0276  06 13             ASL  PTR         ; FA SCORRERE PTR 3 VOLTE
                                                ; A SINISTRA - MOLTIPLICA
                                                ; PER 8.
0101   0278  26 14             ROL  PTR+1
0102   027A  06 13             ASL  PTR
0103   027C  26 14             ROL  PTR+1
0104   027E  06 13             ASL  PTR
0105   0280  26 14             ROL  PTR+1
0106   0282  18                CLC
0107   0283  A5 10             LDA  TABLE       ; SOMMA PUNTATORE ED
                                                ; INDIRIZZO INIZIO TABELLA
0108   0285  65 13             ADC  PTR         ; E POSIZIONA IL RISULTATO
                                                ; NEL PUNTATORE.
0109   0287  85 13             STA  PTR
0110   0289  A5 11             LDA  TABLE+1
0111   028B  65 14             ADC  PTR+1
0112   028D  85 14             STA  PTR+1
0113   028F  60                RTS
0114   0290                    ;
0115   0290                    ; ROUTINE PER GENERARE L'INDICE UNITA' DATI
                               ; IN TABELLA
0116   0290                    ; MEDIANTE CHIAVE HASHING, O CAR. DI LABEL.
0117   0290                    ;
0118   0290  A9 00     HASH    LDA  #0          ; AZZERA LOCAZIONE PER
                                                ; INDICE.
0119   0292  18                CLC              ; PREPARATI ALLA SOMMA.
0120   0293  A0 05             LDY  #5          ; RICICLA 6 VOLTE PER OR
                                                ; ESCLUSIVO.
0121   0295  59 16 00  EXOR    EOR  BUFFER, Y   ; OR ESCLUSIVO DI ACC. CON
                                                ; CARATT. BUFFER.
0122   0298  2A                ROL  A           ; MOLTIPLICA L'ACC. PER 2.
0123   0299  88                DEY              ; CONTA I CARATTERI.
0124   029A  10 F9             BPL  EXOR        ; ACCETTA NUOVO
                                                ; CARATTERE.
0125   029C  85 12             STA  INDX        ; MEMORIZZA IL PRODOTTO
                                                ;  HASH COME INDICE.
0126   029E  60                RTS              ; FATTO.
0127   029F                    END

ERRORS = 000 <000>

SYMBOL TABLE

BAD     025D  BUFFER  0016  CHKLP  0248  CLRLP  0209
CMPR1   0227  CMPR2   0243  DECR   0221  EMPTY  0233
ENTNUM  0015  EXOR    0295  FILL   0235  FIND   023E
HASH    0290  INDX    0012  INIT   0200  LIMIT  0262
HATCH   0252  OK      026E  PTR    0013  SHADD  0271
STORE   0222  TABLE   0010  TEST   0264  XFER   0254

END OF ASSEMBLY
```

Figura 9-45. Programma Hashing

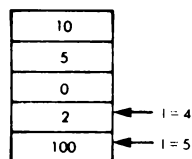
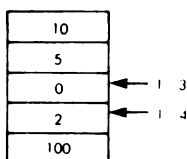
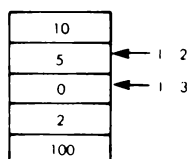
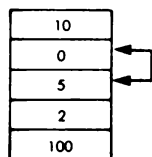
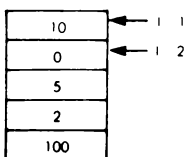
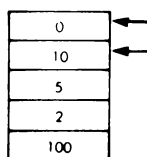
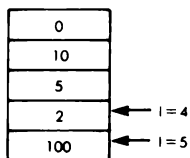
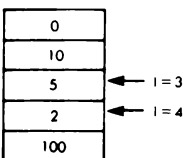
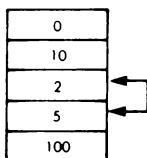
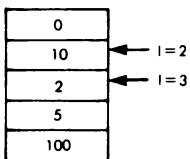
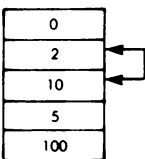
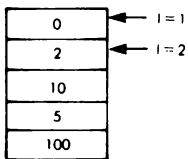



Figura 9.46: Esempio di Bubble-Sort (continua)

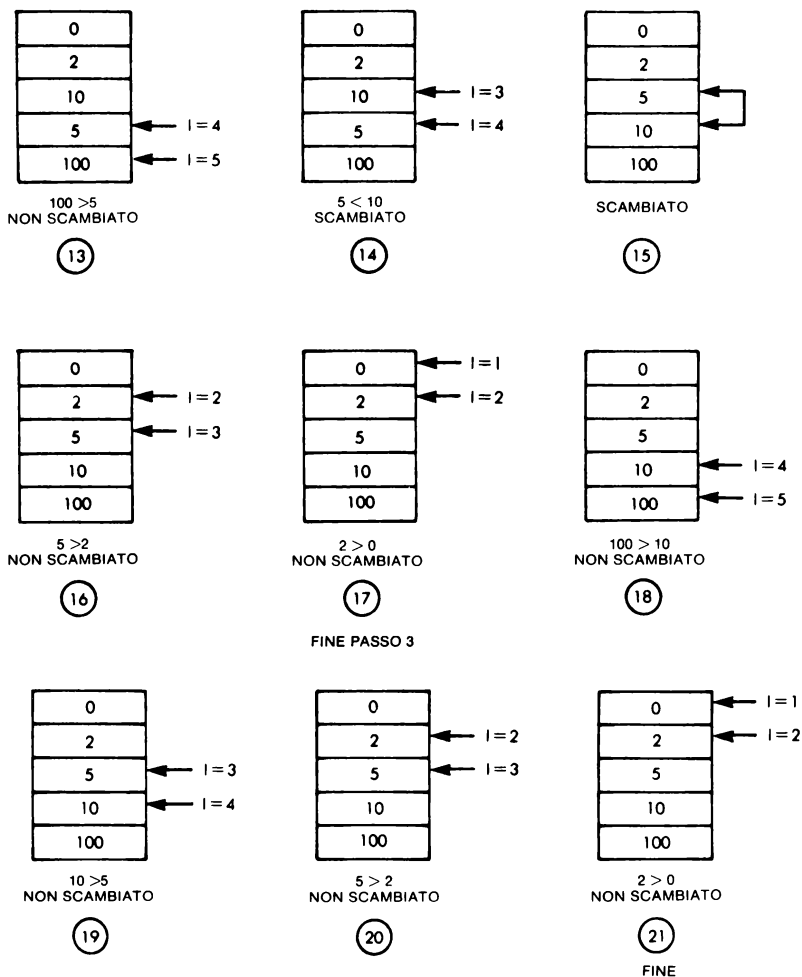



Figura 9.46: Esempio di Bubble-Sort

ricordato sulla coppia di elementi successivi, e così via finchè non sono stati confrontati tutti gli elementi, a due a due.

La Fig. 9-47 illustra il primo passo con le fasi 1, 2, 3, 4, 5 e 6, andando dal basso verso l'alto. (In modo equivalente si potrebbe andare dall'alto al basso).

Se nessun elemento è stato scambiato, la classificazione è completa. Se si è verificato uno scambio, si riparte ancora.

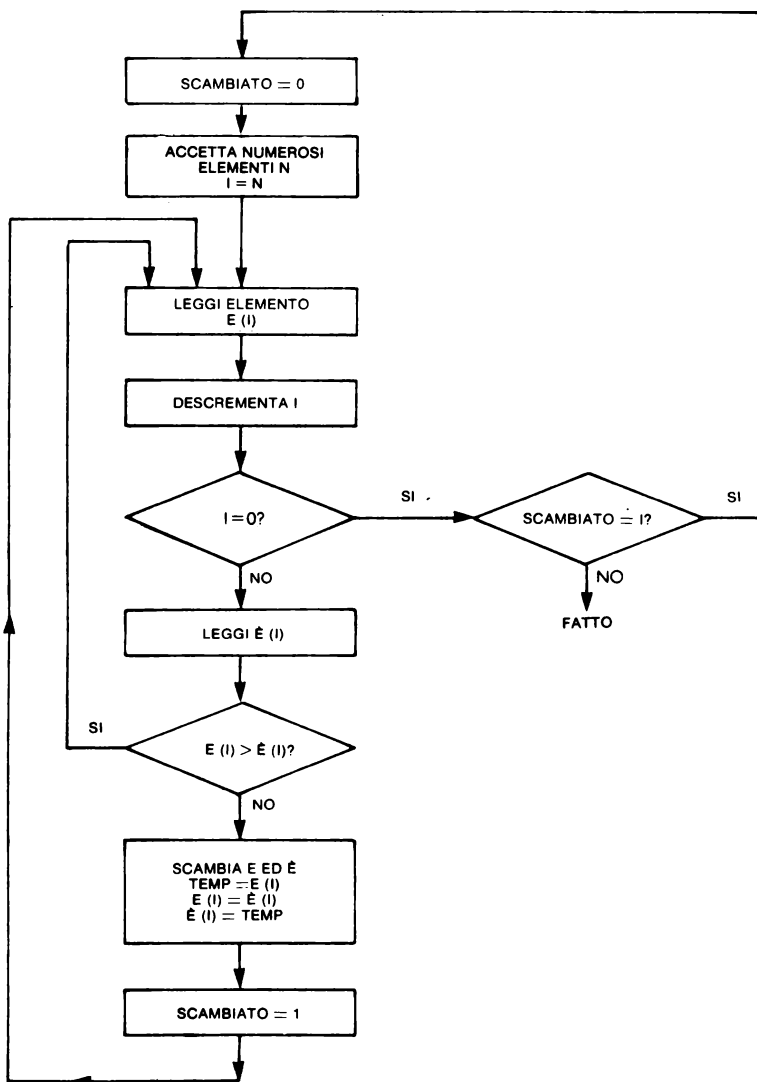



Figura 9.47: Bubble-Sort

In questo semplice esempio, come risulta dalla Fig. 9-47, sono necessari quattro passi.

Il processo descritto è semplice ed impiegato in modo estensivo.

Un'ulteriore complicazione deriva dal meccanismo effettivo di scambio. Scambiando A e B non si può scrivere:

A = B
B = A

poichè ne potrebbe derivare una perdita del valore precedente di A. (Si verifichi con un esempio).

La soluzione corretta è quella di utilizzare una variabile temporanea o una locazione per conservare il valore di A:

```
TEMP  = A
A     = B
B     = TEMP
```

Questo metodo è corretto. (Si verifichi con un esempio). Questa è una cosiddetta permutazione circolare ed è il modo impiegato in tutti i programmi per realizzare gli scambi. La Fig. 9-47 mostra il diagramma di flusso di questa tecnica.

La mappa di memoria corrispondente al programma di bubble-sort è mostrata in Fig. 9-48. In questo programma ogni elemento è un numero

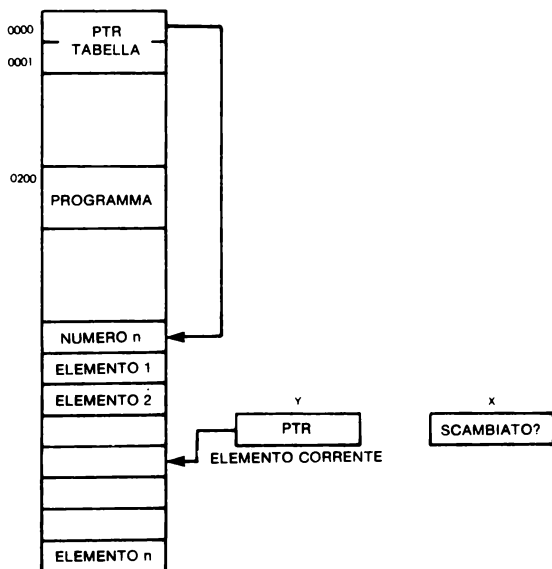



Figura 9.48: Bubble-Sort: mappa di memoria

```
SORT ...... PAGE 0001

LINEA  #LOC  CODICE    LINEA

0002   0000            ;        PROGRAMMA BUBBLE SORT
0003   0000            ;
0004   0000                     * =   $ 0
0005   0000            ;
0006   0000  00 06     TAB      WORD $ 600
0007   0002            ;
0008   0002                     * =   $ 200
0009   0200            ;
0010   0200  A2 00     SORT     LDX   # 0        ; PONI SCAMBIATO A 0.
0011   0202  A1 00              LDA   (TAB,)X
0012   0204  AB                 TAY              ; IL NUMERO DI ELEMENTI È IN
                                                 ; IN Y.
0013   0205  B1 00     LOOP     LDA   (TAB),Y    ; LEGGI L'ELEMENTO E (I).
0014   0207  88                 DEY              ; DECREMENTA IL NUMERO DI
                                                 ; ELEMENTI DA LEGGERE.
0015   0208  F0 12              BEQ   FINISH     ; FINE SE TERMINATI
                                                 ; ELEMENTI.
0016   020A  D1 00              CMP   (TAB),Y    ; CONFRONTA CON E (I).
0017   020C  B0 F7              BCS   LOOP       ; ACCETTA L'ELEMENTO
                                                 ; SUCCESSIVO SE E (I) > È (I).
0018   020E  AA        EXCH     TAX              ; SCAMBIARE GLI ELEMENTI.
                                                 ;
0019   020F  B1 00              LDA   (TAB),Y
0020   0211  CB                 INY
0021   0212  91 00              STA   (TAB),Y
0022   0214  8A                 TXA
0023   0215  88                 DEY
0024   0216  91 00              STA   (TAB),Y.
0025   0218  A2 01              LDX   N1         ; SE È STATO FATTO QUALCHE
                                                 ; SCAMBIO, FA UN ALTRO
                                                 ; PASSO.
0026   021A  D0 E9              BNE   LOOP       ; ACCETTA L'ELEMENTO
                                                 ; SUCCESSIVO.
0027   021C  8A        FINISH   TXA              ; SPOSTA SCAMBIATO NEL
                                                 ; REGISTRO A PER VEDERE ....
0028   021D  D0 E1              BNE   SORT       ; SE È STATO FATTO QUALCHE
0029   021F  60                 RTS              ; SCAMBIO, FA UN ALTRO
                                                 ; PASSO.
0030   0220                     .END.

ERRORS = 0000 < 0000 >
SYMBOL TABLE

SYMBOL   VALUE

EXCH     020E   FINISH   021C   LOOP     0205   SORT     020D.
TAB      0000.
END OF ASSEMBLY
```

Figura 9.49: Programma Bubble-Sort

positivo di 8 bit. Il programma risiede agli indirizzi 200 e successivi. Il registro X viene utilizzato per memorizzare se si è verificato o no uno scambio, mentre il registro Y viene utilizzato come puntatore corrente all'interno della tabella. Si assume che TAB sia l'indirizzo iniziale della tabella. La Fig. 9-49 riporta il programma reale. Per un accesso efficiente

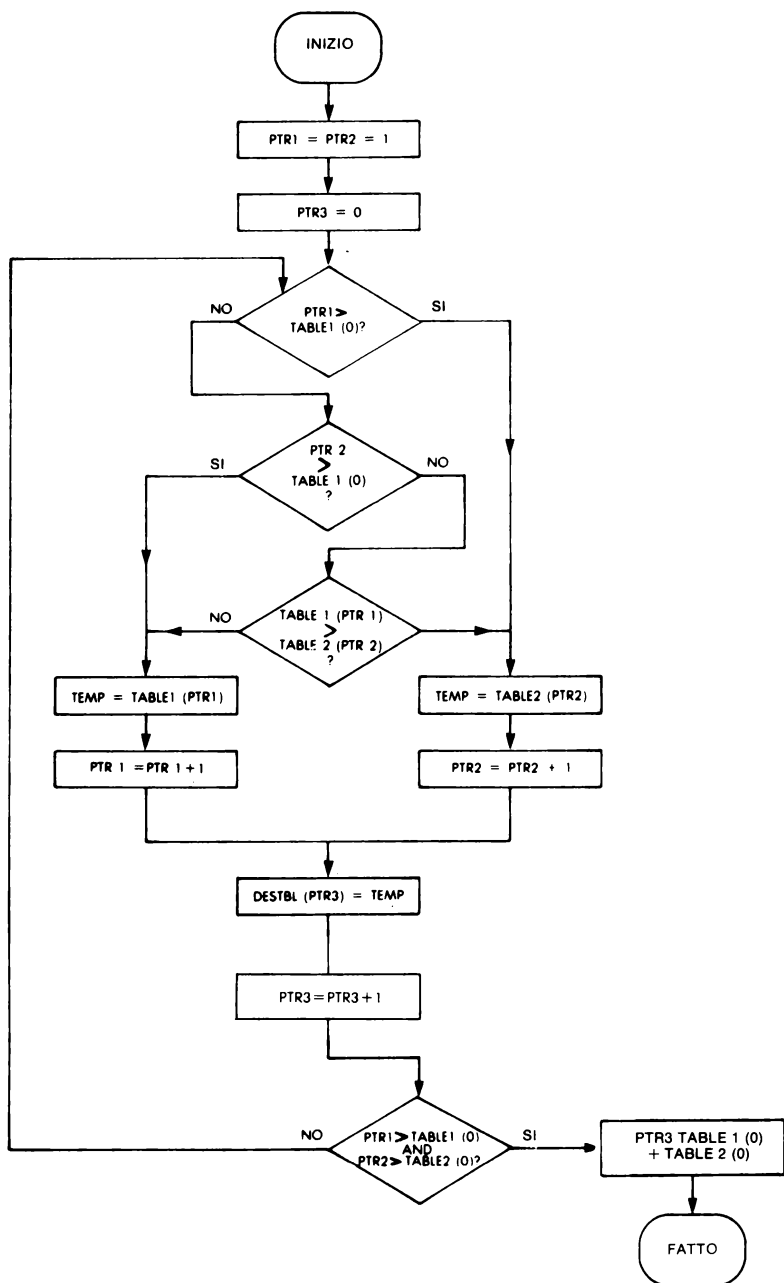



Figura 9.50: Diagramma di flusso Merge

viene utilizzata una tecnica di indirizzamento indiretto indicizzato. Si noti che il programma è molto breve, grazie all'efficienza del modo di indirizzamento indiretto del 6502.

## UN ALGORITMO MERGE

Un altro problema comune consiste nell'unione di due insiemi di dati in un terzo. Si suppone di dover classificare due tabelle di dati e di fonderle in una terza. La lunghezza di ciascuna delle tabelle originali è limitata ad un massimo di 256 byte (una pagina). Il primo ingresso di ogni tabella contiene la lunghezza della stessa.

La Fig. 9-50 mostra l'algoritmo per l'unione di due tabelle. La Fig. 9-51 riporta l'organizzazione di memoria corrispondente e la Fig. 9-52 il programma. Prima di utilizzare il programma è indispensabile posizionare le tabelle "TABLE1", "TABLE2" E "DESTRBL".

L'algoritmo è immediato. Due puntatori correnti, PTR1 e PTR2 puntano alle due tabelle sorgente. PTR3 punta alla tabella risultante.

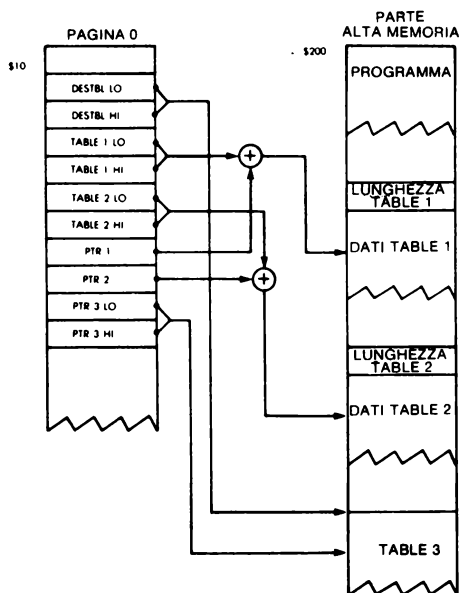



Figura 9.51: Mappa di memoria Merge

```
LINEA  #LOC  CODICE  LINEA

0002   0000          ; MERGE DI 2 PAGINE.
0003   0000          ; PRELEVA 2 TABELLE DATI PRECEDENTEMENTE
0004   0000          ; ORDINATE E LE FONDE IN UNA TERZA TABELLA.
0005   0000          ; OGNI TABELLA SORGENTE PUO' ARRIVARE.
0006   0000          ; AD UNA PAGINA DI LUNGHEZZA (256 BYTE).
0007   0000          ; IL PRIMO ELEMENTO DELLE TABELLE SORGENTE
0008   0000          ; DEVE CONTENERE LA LUNGHEZZA TABELLA.
0009   0000          ; 'PTR3' CONTIENE LA LUNGHEZZA DELLA
0010   0000          ; TABELLA DESTINAZIONE, AL RITORNO.
0011   0000          ;.
0012   0000                  *=     $ 10
0013   0010          DESTBL  *=*+ 2          ; PUNTATORE ALL'INIZIO
                                             ; TABELLA DESTINAZIONE.
0014   0012          TABLE 1 *=*+ 2          ; PUNTATORE ALLA TABELLA
                                             ; SORGENTE 1.
0015   0014          TABLE 2 *=*+ 2          ; PUNTATORE ALLA TABELLA
                                             ; SORGENTE 2.
0016   0016          PTR1    *=*+ 1          ; INDICE TABELLA 1.
0017   0017          PTR2    *=*+ 1          ; INDICE TABELLA 2.
0018   0018          PTR3    *=*+ 2          ; INDICE TABELLA
                                             ; DESTINAZIONE.
0019   001A          ;
0020   001A                  *=*    $ 200.
0021   0200          ;
0022   0200  A5 11           LDA  DESTBL+1   ; PTR3 = TABLE 3.
0023   0202  85 19           STA  PTR3+1.
0024   0204  A5 10           LDA  DESTBL.
0025   0206  85 18           STA  PTR3.
0026   0208  A9 01           LDA  # 1        ; POSIZIONA ALL'INIZIO
                                             ; I PUNTATORI TABELLE
                                             ; SORGENTE,.
0027   020A  85 16           STA  PTR1       ; SALTANDO LE LUNGHEZZE
                                             ; TABELLE.
0028   020C  85 17           STA  PTR2.
0029   020E  A2 00           LDX  # 0        ; AZZERA X PER
                                             ; PER INDIRIZZAMENTO
                                             ; INDIRETTO.
0030   0210  A1 14   COMPR   LDA  (TABLE2,X) ; LA TABELLA 2 HA
                                             ; LUNGHEZZA <.
0031   0212  C5 17           CMP  PTR2       ; PUNTATORE TABELLA 2?
0032   0214  90 19           BCC  TKTB1      ; SE SI, ACCETTA BYTE DA
                                             ; TABELLA 1.
0033   0216  A1 12           LDA  (TABLE 1,X) ; LA LUNGHEZZA TABELLA 1
                                             ; È <
0034   0218  C5 16           CMP  PTR1       ; PUNTATORE TABELLA 1?
0035   021A  90 0A           BCC  TKTB2      ; SE SI, ACCETTA BYTE DA
                                             ; TABELLA 2.
0036   021C  A4 16           LDY  PTR1       ; ACCETTA PUNTATORE
                                             ; TABELLA 1.
0037   021E  B1 12           LDA  (TABLE1),Y ; UTILIZZALO PER PRELEVARE
                                             ; BYTE.
0038   0220  A4 17           LDY  PTR2       ; ACCETTA PUNTATORE
                                             ; TABELLA 2.
0039   0222  B1 14           CMP  (TABLE2),Y ; UTILIZZALO PER TROVARE
                                             ; BYTE DA.
0040   0224                  ; CONFRONTARE CON BYTE TABELLA 1.
0041   0224  90 09           BCC  TKTB1      ; SE BYTE TABELLA 1 È MINORE
                                             ; PRENDILO.
0042   0226  A4 17   TKTB2   LDY  PTR2       ; ACCETTA PUNTATORE
                                             ; TABELLA 2.
```

Figura 9-52. Programma Merge (continua)

```
0043   0228  B1 14             LDA   (TABLE2),Y  ; ACCETTA BYTE SUCCESSIVO
                                                 ; DA TABELLA 2.
0044   022A  E6 17             INC   PTR2        ; INCREMENTA PUNTATORE
                                                 ; TABELLA 2.
0045   022C  4C 35 02          JMP   STORE       ; MEMORIZZA IL BYTE NELLA
                                                 ; TABELLA DI DESTINAZIONE.
0046   022F  A4 16    TKTB1    LDY   PTR1        ; ACCETTA PUNTATORE 1.
0047   0231  B1 12             LDA   (TABLE1),Y  ; ED UTILIZZALO PER
                                                 ; PRELEVARE BYTE DALLA
                                                 ; TABELLA.
0048   0233  E6 16             INC   PTR1        ; INCREMENTA PUNTATORE
                                                 ; TABELLA 1.
0049   0235  81 18    STORE    STA   (PTR3,X)    ; MEMORIZZA IL BYTE ALLA
                                                 ; LOCAZIONE SUCCESSIVA IN
                                                 ; TABELLA 3.
0050   0237  E6 18             INC   PTR3        ; INCREMENTA IL PUNTATORE
                                                 ; DI BASSO ORDINE DELLA
                                                 ; TABELLA 3.
0051   0239  D0 02             BNE   CC          ; SE NON OVERFLOW, SALTA.
0052   023B  E6 19             INC   ; PTR3+3
                                                 ; INCREMENTA IL PUNATORE
                                                 ; DI ORDINE ELEVATO DELLA
                                                 ; TABELLA 2.
0053   023D  A1 12    CC       LDA (TABLE 1,X)   ;LA LUNGHEZZA DELLA
                                                 ; TABELLA 1 È MAGGIORE
0054   023F  C5 16             CMP   PTR1        ; O UGUALE A PUNTATORE 1?
0055   0241  B0 CD             BCS   COMPR       ; SE SI ACCETTA BYTE
                                                 ; SUCCESSIVO.
0056   0243  A1 14             LDA   (TABLE2,X)  ; LA LUNGHEZZA TABELLA 2 È
                                                 ; MAGGIORE.
0057   0245  C5 17             CMP   PTR2        ; O UGUALE AL PUNTATORE 2?
0058   0247  B0 C7             BCS   COMPR       ; SE SI, ACCETTA BYTE
                                                 ; SUCCESSIVO.
0059   0249  A9 00             LDA   # 0
0060   024B  85 19             STA   PTR3+1      ; AZZERA PTR3 DI ORDINE
                                                 ; ELEVATO
0061   024D  18                CLC               ; MERGE ESEGUITO, ORA ...
0062   024E  A1 12             LDA   (TABLE1,X)  ; SOMMA LE LUNGHEZZE
                                                 ; DELLE TABELLE 1 E 2.
0063   0250  61 14             ADC   (TABLE2,X)
0064   0252  85 18             STA   PTR3        ; MEMORIZZA LA SOMMA NEL
                                                 ; PUNTATORE TEMPORANEO
                                                 ; TABELLA 3.
0065   0254  90 04             BCC   CCC         ; ED ...
0066   0256  A9 01             LDA   # 1         ; OVERFLOW IN
0067   0258  85 19             STA   PTR3+1      ; BYTE DI ORDINE ELEVATO.
0068   025A  60       CCC      RTS
0069   025B                    .END

ERRORS = 0000 < 0000 >
END OF ASSEMBLY
```

Figura 9-52. Programma Merge

Gli ingressi correnti di TABLE 1 e TABLE 2 sono confrontati due alla volta. Quello più piccolo viene copiato in TABLE3 ed il puntatore corrente viene incrementato. Il processo viene ripetuto e termina quando PTR1 e PTPR2 hanno raggiunto il fondo delle rispettive tabelle.

## SOMMARIO

Sono stati presentati gli esempi reali di realizzazione ed i concetti di base relativi alle strutture dati pù comuni.

Il 6502, grazie ai suoi potenti modi di indirizzamento, consente la manipolazione di struture dati complesse. La sua efficienza è dimostrata dalla semplicità dei programmi mostrati.

Inoltre sono state presentate delle tecniche speciali per l'hashing, sorting e merging, tipicamente utilizzate per la risoluzione di problemi complessi relativi alle strutture dati.

Il programmatore principiante non deve preoccuparsi per i dettagli della realizzazione e manipolazione di strutture dati. Comunque per una programmazione efficiente di algoritmi non banali, è indispensabile una buona conoscenza delle strutture dati. Gli esempi reali presentati in questo capitolo possono aiutare tutti i problemi comuni che si incontrano nelle strutture dati reali.

CAPITOLO 10

# SVILUPPO DEL PROGRAMMA

## INTRODUZIONE

Tutti i programmi studiati e sviluppati finora sono stati sviluppati a mano senza l'aiuto di qualsiasi risorsa software oppure hardware. Il solo miglioramento che è stato utilizzato rispetto alla codifica binaria diretta è stato l'impiego dei simboli mnemonici del linguaggio assembly. Per l'effettivo sviluppo software è necessario capire la gamma di aiuti dello sviluppo software ed hardware. Questo capitolo si propone di presentare e valutare questi aiuti.

## SCELTE DI BASE DELLA PROGRAMMAZIONE

Esistono tre alternative di base: scrittura di un programma in binario od esadecimale, scrittura in linguaggio di livello assembly oppure scrittura in linguaggio ad alto livello. Si analizzeranno queste alternative.

### 1. Codifica Esadecimale

Il programma sarà normalmente scritto utilizzando i mnemonici in linguaggio assembly. Comunque i sistemi calcolatori a scheda singola, di costo più basso non sono forniti di un assemblatore. L'assemblatore è il programmatore che opera la traduzione automatica dei mnemonici utilizzati per il programma nei codici binari richiesti. Quando non è disponibile un assemblatore questa traduzione da mnemonici in binario deve essere eseguita a mano. Il binario è spiacevole e genera facilmente errori cosicchè viene utilizzato normalmente l'esadecimale. È stato mostrato al Capitolo 1 che un digit esadecimale rappresenta 4 bit binari. Due digit esadecimali saranno perciò utilizzati per rappresentare i contenuti di ciascun byte. Come esempio viene riportata in Appendice la tabella che mostra l'equivalente esadecimale delle istruzioni del 6502.

In breve ogni volta che le risorse dell'utente sono limitate e non è disponibile l'assemblatore occorrerà tradurre manualmente il programma in esadecimale. Questo può essere fatto ragionevolmente per un

piccolo numero di istruzioni per esempio da 10 a 100. Per programmi più lunghi questo processo è tedioso e predisposto agli errori cosicchè esso tende a non essere utilizzato. Comunque quasi tutti i microcalcolatori su scheda singola richiedono l'ingresso dei programmi in modo esadecimale. Essi non sono equipaggiati di un assemblatore e di un'intera tastiera alfanumericca in modo da limitare il loro costo.

In conclusione la codifica esadecimale non è un modo desiderabile per accedere in un calcolatore. Esso è semplicemente un modo economico. Il costo di un assemblatore e della tastiera alfanumerica corrispondente è il compromesso con l'aumento di lavoro per far entrare il programma nella memoria. Comunque questo non cambia il modo in cui è scritto il programma stesso. *Il programma viene ancora scritto in linguaggio di livello assembly* cosicchè esso possa essere ispezionato ed esaminato dal programmatore umano ed essere significativo.

## 2. Programmazione in Linguaggio Assembly

La programmazione di livello assembly copre sia i programmi che possono entrare nel sistema in forma esadecimale sia quelli che possono entrare in forma simbolica di livello assembly. Si esaminerà ora l'ingresso di un programma direttamente nella sua rappresentazione in linguaggio assembly. Deve essere disponibile un programma assemblatore. L'assemblatore leggerà ciascuna istruzione mnemonica del programma e la tradurrà nello schema di bit richiesto utilizzando 1, 2 oppure 3 byte, secondo quanto specificato dalla codifica delle istruzioni. Inoltre un buon assemblatore offrirà un certo numero di possibilità addizionali per la scrittura del programma. Questo sarà analizzato in seguito nel paragrafo relativo all'assemblatore. In particolare sono disponibili le *direttive* che modificheranno il valore dei simboli. Può essere utilizzato l'indirizzamento simbolico e può essere specificata una diramazione dalla locazione simbolica. Durante la fase di collaudo, dove un utente può rimuovere oppure aggiungere istruzioni, non sarà necessario riscrivere l'intero programma se un'ulteriore istruzione viene inserita tra una diramazione ed il punto in cui essa opera la diramazione, utilizzando label simboliche.
L'assemblatore si occuperà di aggiustare automaticamente tutte le label durante il processo di traduzione. Inoltre un assemblatore consente all'utente di collaudare il suo programma in forma simbolica. Un disassemblatore può essere utilizzato per esaminare i contenuti della locazione di memoria e ricostruire l'istruzione di livello assembly che essa rappresenta. Verranno analizzate di seguito le varie risorse software

normalmente disponibili su un sistema. Si esamini ora la terza alternativa.

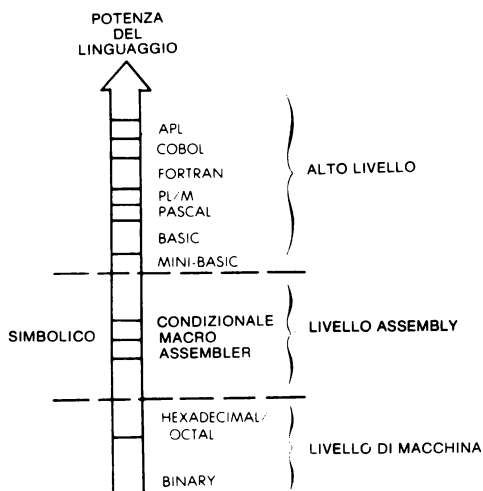



Figura 10.1: Livelli di programmazione

### 3. Linguaggio ad Alto Livello

Un programma può essere scritto in un programma ad alto livello come BASIC, APL, PASCAL od altri. Le tecniche di programmazione in questi vari linguaggi sono coperte da libri specifici e non saranno analizzate in questa sede. Perciò si analizzerà soltanto questo modo di programmazione. Un linguaggio ad alto livello offre istruzioni potenti che rendono pù facile e veloce la programmazione. Queste istruzioni devono essere tradotte da un programma complesso nella rappresentazione binaria finale che un microcalcolatore può eseguire. Tipicamente ciascuna istruzione ad alto livello sarà tradotta in un gran numero di istruzioni binarie singole. Il programma che eseguirà questa traduzione automatica è chiamato un *compilatore* ovvero un *interprete.* Un compilatore tradurrà tutte le istruzioni di un programma in sequenza in codice oggetto. In una fase separata il codice risultante sarà quindi eseguito. Per contrasto un interprete interpreterà ed eseguirà una singola istruzione, quindi "tradurrà" quella successiva. Un interprete ha il vantaggio responso interattivo, ma ha una bassa efficienza rispetto al compilatore. In questa sede non si entrerà in ulteriori dettagli ma si considera la programmazione in linguaggio di livello assembly di un microprocessore reale.

## SUPPORTO SOFTWARE

Si analizzeranno ora le principali caratteristiche software disponibili (o che dovrebbero essere disponibili) in un sistema completo di sviluppo software conveniente. Alcune definizioni sono già state introdotte. Queste saranno riassunte e prima di procedere saranno definiti i rimanenti programmi importanti.

*L'assemblatore* è il programma che traduce la rappresentazione mnemonica delle istruzioni nel loro equivalente binario. Esso normalmente traduce un'istruzione simbolica in un'istruzione binaria (che può occupare 1, 2 oppure 3 byte). Il codice binario risultante è chiamato *codice oggetto.* Esso è direttamente eseguibile dal microcalcolatore. Come effetto secondario l'assemblatore produrrà anche una lista simbolica completa del programma come le tabelle di equivalenza da utilizzare da parte del programmatore e la lista dei simboli occorrenti nel programma. Gli esempi saranno presenti in seguito nel corso del capitolo.

Un *compilatore* è il programma che traduce le istruzioni ad alto livello nella loro forma binaria.

Un *interprete* è il programma che traduce, analogamente al compilatore, le istruzioni ad alto livello nella loro forma binaria ma non conserva la rappresentazione intermedia e si ha l'esecuzione immediata. Infatti spesso non si ha addirittura la generazione di qualsiasi codice intermedio ma piuttosto esso esegue direttamente le istruzioni ad alto livello.

Un *monitor* è il programma di base che è indispensabile per utilizzare le risorse hardware di questo sistema. Esso osserva continuamente i dispositivi d'ingresso per l'ingresso e dirige il resto dei dispositivi. Per esempio un monitor minimo per un microcalcolatore su scheda singola, equipaggiato di tastiera a LED, deve esplorare continuamente la tastiera come ingresso utente e mostrare i contenuti specifici sui diodi-emettitori-di-luce. Inoltre esso deve essere in grado di riconoscere un certo numero di comandi limitati dalla tastiera, come START, STOP, CONTINUA, CARICA MEMORIA, ESAMINA MEMORIA. Sui grossi sistemi il monitor è spesso qualificato come programma *esecutivo.* In questo caso è disponibile la direzione del file complesso ovvero la gestione di schedari. Il set globale delle caratteristiche è detto *sistema operativo.* Nel caso in cui i file possono essere residenti su disco, il sistema operativo è qualificato come *sistema operativo su disco ovvero DOS.*

Un *editor* è il programma progettato per consentire l'ingresso e la modifica del testo o dei programmi. Esso consente all'utente di far entrare convenientemente i caratteri, agganciarli, inserirli, aggiungere righe, rimuovere righe, ricercare caratteri o stringhe. Questa è una risorsa importante per ingresso coveniente.

Un *debugger* è una caratteristica necessaria per collaudare i programmi. Ogni volta che un programma non lavora correttamente tipicamente può non esserci indicazione della causa, qualunque essa sia.
Il programmatore perciò desidera inserire dei punti di arresto nel suo programma in modo da sospendere l'esecuzione del programma agli indirizzi specificati ed essere in grado di esaminare i contenuti dei registri e della memoria in questi punti. Il debugger consente la sospensione di un programma, la ripresa dell'esecuzione, l'esame, l'osservazione e la modifica dei contenuti dei registri o della memoria. Un buon debugger sarà equipaggiato di un certo numero di caratteristiche addizionali come la possibilità di esaminare i dati in forma simbolica, esadecimale, binaria od altre rappresentazioni usuali come pure l'ingresso dei dati in questo formato.

Un *caricatore*, ovvero *caricatore di collegamento* posizionerà i vari blocchi in codice oggetto alle posizioni specificate nella memoria ed aggiusta i rispettivi puntatori simbolici cosicchè si possa far loro riferimento. Esso è utilizzato per la rilocazione di programmi o blocchi in diverse aree della memoria.

Un programma *simulatore* od *emulatore* è utilizzato per simulare il funzionamento di un dispositivo, normalmente il microprocessore, in sua assenza, quando si sta sviluppando un programma su un processore simulato prima di posizionarlo sulla scheda effettiva. Utilizzando questo approccio diviene possibile sospendere il programma, modificarlo e conservarlo in una memoria RAM. Gli svantaggi di un simulatore sono:

1. Esso normalmente simula soltanto il processore stesso e non i dispositivi d'ingresso/uscita.
2. La velocità di esecuzione è bassa e si opera in tempo simulato. Non è perciò possibile provare dispositivi in tempo reale e possono verificarsi problemi di sincronizzazione anche se la logica del programma può essere rilevata corretta.

Un *emulatore* è essenzialmente un simulatore in tempo reale. Esso utilizza un processore per simularne un altro e lo simula completamente sino ai dettagli.

Le *Utility routines* sono essenzialmente tutte le routine normalmente necessarie nella maggior parte delle applicazioni e che l'utente desidera gli vengano fornite dal costruttore! Esse possono comprendere la moltiplicazione, la divisione ed altre operazioni aritmetiche, routine di movimento di blocco, verifiche di carattere, manipolazioni di dispositivi d'ingresso/uscita (ovvero "driver") ed altre.

## LA SEQUENZA DI SVILUPPO DEL PROGRAMMA

Si esaminerà ora una sequenza tipica di sviluppo di un programma di livello assembly. Si assumerà che tutte le caratteristiche software usuali siano disponibili in modo da dimostrare il loro valore. Se queste dovessero non essere disponibili in un sistema particolare sarà ancora possibile sviluppare i programmi ma la convenienza diminuirà e, conseguentemente, è probabile che il tempo necessario per il collaudo del programma sia destinato ad aumentare.

L'approccio più comune consiste innanzitutto nel progetto dell'algoritmo e nella definizione delle strutture dati appropriati al problema da risolvere. Successivamente occorre sviluppare un insieme completo di diagrammi di flusso che rappresentano il flusso del programma. Infine i diagrammi di flusso sono tradotti nel linguaggio di livello assembly del microprocessore: questa è la fase di codifica.

In seguito il programma viene fatto entrare nel calcolatore. Si esamineranno al paragrafo successivo le scelte hardware da utilizzare in questa fase.

Il programma è fatto entrare nella memoria RAM del sistema sotto il controllo dell'editor. Una volta che è entrata una sezione del programma, per esempio una o più subroutine, essa sarà verificata.
Innanzitutto si userà l'assemblatore. Se l'assemblatore non risiede già nel sistema esso verrà caricato da una memoria esterna, come un disco. Quindi il programa sarà assemblato, cioè tradotto in un codice binario. Questo fa in modo che il programma oggetto sia pronto per essere eseguito.

Normalmente non ci si deve aspettare che un programma lavori correttamente la prima volta. Per verificare il suo funzionamento corretto occorrerà posizionare in locazioni cruciali un certo numero di punti di arresto dove è facile verificare se i risultati intermedi sono corretti. Il debugger sarà utilizzato per questo scopo. I punti di arresto saranno specificati in locazioni selezionate. Verrà quindi emesso un

comando "Go" cosicchè venga iniziata l'esecuzione del programma. Il programma si arresterà automaticamente ad ogni punto di arresto specificato. Il programmatore può quindi verificare, esaminando i contenuti dei registri, o della memoria, che i dati ottenuti siano corretti. Se questo si verifica si procede fino al punto di arresto successivo. Ogni volta che si trova un dato non corretto è presente un errore nel programma. A questo punto normalmente il programmatore fa riferimento alla lista del programma e verifica se la sua codifica è stata eseguita correttamente. Se non si riesce a trovare nessun errore nella programmazione, l'errore deve essere logico e si deve fare riferimento al diagramma di flusso. Qui si assumerà che i diagrammi di flusso siano stati controllati a mano e che si riterranno ragionevolmente corretti. L'errore probabilmente può provenire dalla codifica. Sarà perciò necessario modificare una parte del programma. Se la rappresentazione simbolica del programma è ancora nella memoria, si farà semplicemente rientrare l'editor e si modificheranno le linee richieste e quindi si ripeterà ancora la sequenza precedente. In alcuni sistemi la memoria disponibile può non essere grande abbastanza, cosicchè è necessario far uscire la rappresentazione simbolica del programma su un disco o cassetta prima dell'esecuzione del codice oggetto. naturalmente in questo caso si dovrebbe ricaricare la rappresentazione simbolica del programma dal suo mezzo di supporto prima del rientro dell'editor.

La procedura precedente sarà ripetuta necessariamente finchè i risultati del programma sono corretti. Si sottolinea che la prevenzione è molto più efficiente della cura. Un progetto corretto si risolverà tipicamente in un programma che opera correttamente e molto velocemente una volta che gli errori più comuni ed ovvi di codifica sono stati rimossi.

Comunque un progetto confuso può risolversi in programmi che impiegheranno un tempo estremamente lungo per essere collaudato. Il tempo di collaudo è generalmente considerato essere molto più lungo dell'effettivo tempo di progetto. In breve vale sempre la pena impiegare più tempo nel progetto in modo da abbreviare la fase di collaudo.

Comunque, impiegando questo approccio, è possibile verificare l'organizzazione globale del programma ma non verificarlo in tempo reale con i dispositivi d'ingresso/uscita. Se devono essere verificati i dispositivi d'ingresso/uscita la soluzione diretta consiste nel trasferimento del programma in EPROM nella sua installazione su scheda e quindi nell'osservazione se esso lavora.

Esiste una soluzione migliore. È l'impiego di un *emulatore in circuito.* Un emulatore in circuito utilizza il microproccessore 6502 (o qualsiasi

altro) per simulare un 6502 (quasi) in tempo reale. Esso simula fisicamente il 6502. L'emulatore è equipaggiato con un cavo terminante in un connettore a 40 pin esattamente identico ai pin di uscita del 6502. Questo connettore può poi essere inserito sulla scheda di applicazione effettiva che si sta sviluppando. I segnali generati dall'emulatore saranno esattamente quelli del 6502, forse soltanto un pò più lenti. Il vantaggio essenziale è che il programma che si sta verificando risiederà ancora nella memoria RAM del sistema di sviluppo. Esso genererà i segnali effettivi che comunicheranno con i dispositivi d'ingresso/uscita effettivi che si desidera utilizzare. Ne risulta che diviene possibile eseguire lo sviluppo

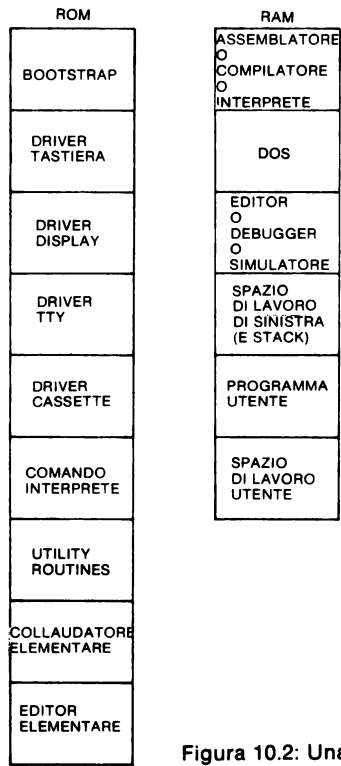



Figura 10.2: Una mappa di memoria tipica

del programma utilizzando tutte le risorse del sistema di sviluppo (editor, debugger, caratteristiche simboliche, sistema file) mentre si sta verificando l'ingresso/uscita in tempo reale.

Inoltre un buon emulatore fornisce caratteristiche speciali, come un *trace.* Un trace è una registrazione delle ultime istruzioni o dello stato dei vari bus dati del sistema prima di un punto di arresto. In breve un trace fornisce la sequenza di eventi che si verificano prima di un punto di arresto o di un malfunzionamento. Esso può anche far scattare uno scope all'indirizzo specificato oppure, all'occorrenza, ad una specificata combinazione di bit. Una tale caratteristica è di grande valore poichè quando si trova un errore è normalmente troppo tardi. L'istruzione, od il dato, che ha causato l'errore si è verificato prima della rilevazione. La disponibilità di un trace consente all'utente di trovare quale segmento del programma origina l'errore. Se il trace non è abbastanza lungo si porrà semplicemente prima un punto di arresto.

Questo completa la descrizione della sequenza usuale di eventi coinvolti nello sviluppo di un programma. Si analizzeranno ora le alternative hardware disponibili per sviluppare i programmi.

## LE ALTERNATIVE HARDWARE

### 1. Microcomputer su Scheda Singola

Il microcomputer su scheda singola offre l'approccio di costo più basso allo sviluppo del programma. Esso è normalmente equipaggiato di una tastiera esadecimale, più alcuni tasti di funzione, più 6 LED che possono mostrare indirizzi e dati. Poichè esso è equipaggiato di una piccola quantità di memoria normalmente non è disponibile nessun assembler. Al massimo esso ha un piccolo monitor e virtualmente non ha caratteristiche di editing o debugging eccetto un numero molto limitato di comandi. Tutti i programmi devono entrare perciò in forma esadecimale. Quindi essi saranno mostrati sui LED in forma esadecimale. Un microcomputer su scheda singola ha, in teoria, la stessa potenza hardware di qualsiasi altro calcolatore. Semplicemente a causa della sua dimensione ristretta di memoria e di tastiera esso non soddisfa tutte le caratteristiche di un sistema più grosso e rende lo sviluppo del programma molto più lungo. Poichè è tedioso sviluppare programmi in formato esadecimale, un microcalcolatore su singola scheda è più adatto per l'educazione ed il training dove devono essere sviluppati dei programmi di lunghezza limitata e la loro breve lunghezza non è un ostacolo alla programmazione. Le singole schede costituiscono probabilmente il modo più a buon mercato per imparare eseguendo la programmazione. Comunque esse non possono essere utilizzate per lo sviluppo di programmi complessi senza la connessione di schede di memoria e la disponibilità degli usuali aiuti software.

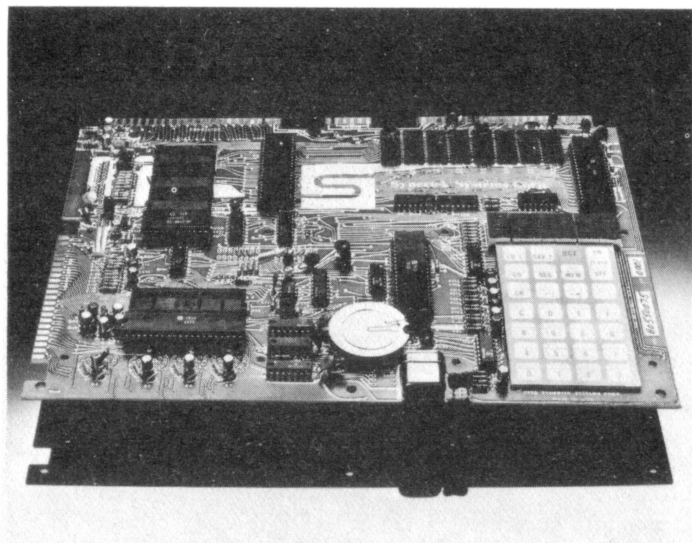



Figura 10.3: Il SYM è una tipica scheda microcomputer

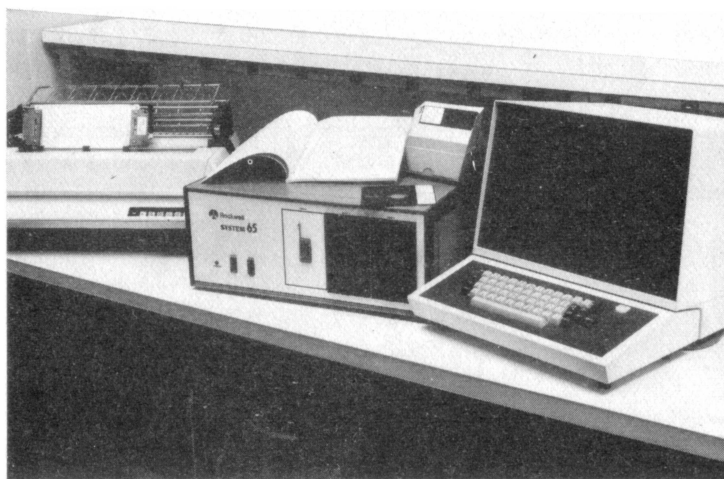



Figura 10.4: Il System 65 Rockwell/Mostek è un sistema di sviluppo

## 2. Il Sistema di Sviluppo

Un sistema di sviluppo è un sistema microcomputer equipaggiato con una quantità significativa di memoria RAM (32K, 48K) come richiesto dai dispositivi d'ingresso/uscita, come un display CRT, una stampante, dischi e normalmente un programmatore PROM come pure, forse, un emulatore in circuito. Un sistema di sviluppo è progettato specificamente per facilitare lo sviluppo del programma in un ambiente industriale.
Esso offre normalmente tutte o quasi tutte le caratteristiche software considerate al paragrafo precedente. In linea di principio esso è lo strumento ideale di sviluppo software.

La limitazione di un sistema di sviluppo di microcomputer è di non essere in grado di sostenere un compilatore oppure un interprete. Questo perchè un compilatore richiede una grande quantità di memoria, spesso molta di più di quella disponibile sul sistema. Comunque per lo sviluppo dei programmi in linguaggio di livello assembly esso offre tutte le caratteristiche richieste. In ogni caso, poichè i sistemi di sviluppo vengono venduti in numero relativamente piccolo rispetto ai computer tipo hobby, il loro costo è significativamente più elevato.

## 3. Microcomputer Tipo Hobby

L'hardware del microcomputer tipo hobby è naturalmente esattamente analogo a quello di un sistema di sviluppo. La principale differenza risiede nel fatto che questo non è normalmente equipaggiato con i sofisticati aiuti di sviluppo software che sono disponibili su un sistema di sviluppo industriale. Per esempio, molti microcomputer tipo hobby offrono solo assemblatori elementari, editor minimi, sistemi di file minimi, assenza di caratteristiche di connessione di un programmatore PROM, assenza di emulatori in circuito, assenza di debugger potenti. Essi rappresentano perciò una fase intermedia tra il microcomputer su singola scheda ed un sistema di sviluppo a microprocessore completo. Per un utente che desidera sviluppare programmi di modesta complessità, essi sono probabilmente il miglior compromesso poichè essi offrono il vantaggio di basso costo ed un ragionevole insieme di strumenti di sviluppo software, anche se essi sono abbastanza limitati rispetto alla loro convenienza.

## 4. Sistema a Divisione di Tempo (Time Sharing)

È possibile affittare terminali da diverse compagnie che li colleghe-

ranno a reti a divisione di tempo. Questi terminali dividono il tempo di un computer più grosso e beneficiano di tutti i vantaggi di una grossa installazione. Sono così disponibili *assemblatori incrociati* per tutti i microcomputer su virtualmente tutti i sistemi commerciali a divisione di tempo. Un assemblatore incrociato è semplicemente un assemblatore, diciamo un 6502, che risiede per esempio su un IBM 370. Formalmente un assemblatore incrociato è un assemblatore per il microprocessore X che risiede sul microprocessore Y. La natura del computer utilizzato è irrilevante. L'utente scrive ancora un programma in linguaggio di livello assembly del 6502 e l'assemblatore incrociato lo traduce nell'appropriata struttura di bit binari. La differenza comunque è che questo programma non può essere eseguito a questo punto. Esso può essere eseguito da un processore simulato, se è disponibile, fornito il quale non si utilizza nessuna risorsa d'ingresso/uscita. Questa soluzione viene perciò utilizzata soltanto in ambienti industriali.

**5. In-House Computer**

Ogni volta che è disponibile un grosso in-house computer, possono essere disponibili anche assemblatori incrociati che facilitano lo sviluppo del programma. Se tale computer offre il servizio di divisione di tempo questa scelta diventa esattamente uguale a quella del paragrafo precedente. Se esso offre solo servizio collettivo questo è probabilmente uno dei metodi più sconvenienti di sviluppo del programma poichè la sottoposizione di programmi in modo collettivo al livello assembly di un microprocessore si risolve in un tempo di sviluppo molto lungo.

## Pannello Frontale oppure Assenza di Pannello Frontale?

Il pannello frontale è un accessorio hardware spesso utilizzato per facilitare il collaudo del programma. Esso è stato uno strumento tradizionale per mostrare i contenuti binari di un registro o della memoria, in modo conveniente. Comunque tutte le funzioni del pannello di controllo possono essere eseguite da un terminale e la predominanza di display CRT ora offre un servizio pressochè equivalente al pannello di controllo mostrando il valore binario dei bit. Il vantaggio ulteriore dell'impiego del display CRT è che si può commutare a volontà dalla rappresentazione binaria a quella esadecimale, simbolica, decimale (naturalmente se sono disponibili le appropriate routine di conversione). Lo svantaggio di un CRT è che si devono premere diversi tasti per ottenere il display appropriato invece di commutare una manopola. Comunque, poichè il costo della fornitura del pannello di controllo è abbastanza sostanziale, la maggior parte dei microprocessori recenti ha abbandonato questo

strumento di collaudo. Il valore del pannello di controllo è spesso valutato più in funzione di argomenti emozionali basati sulla precedente esperienza piuttosto che da una scelta razionale. Questo non è indispensabile.

## SOMMARIO DELLE RISORSE HARDWARE

Si possono distinguere tre grandi casi: se si ha soltanto un budget minimo e se si desidera imparare a programmare è il caso di acquistare un microcomputer su scheda singola. Utilizzando questo si sarà in grado di sviluppare tutti i semplici programmi di questo libro e molto di più. Eventualmente, quando si vogliono sviluppare programmi di più di un centinaio di istruzioni, si risentiranno le limitazioni di questo approccio.

Invece un utente industriale necessita di un sistema di sviluppo completo. Qualsiasi soluzione abbreviata di un sistema di sviluppo completo causerà un tempo di sviluppo significativamente più lungo. Il compromesso è chiaro: risorse hardware rispetto al tempo di programmazione. Naturalmente se i programmi da sviluppare sono abbastanza semplici può essere utilizzato un approccio meno dispendioso. In ogni caso, se si devono sviluppare programmi complessi è difficile giustificare qualsiasi risparmio hardware nell'acquisto di un sistema di sviluppo poichè i costi di programmazione saranno di gran lunga il costo dominante del progetto.

Per impieghi personal computer un microcomputer tipo hobby offrirà caratteristiche tipicamente sufficienti, anche se minime. La maggior parte dei computer tipo hobby non è ancora dotata di un buon software di sviluppo. L'utente dovrà valutare il suo sistema in relazione ai commenti presentati in questo capitolo.

Si analizzerà ora in maggior dettaglio la risorsa più indispensabile: l'assemblatore.

## L'ASSEMBLATORE

Nel corso di questo libro si è utilizzato il linguaggio di livello assembly senza presentare la sintassi formale ovvero la definizione del linguaggio di livello assembly. È ora il momento di presentare queste definizioni. Un assemblatore è progettato per consentire la rappresentazione simbolica conveniente del programma utente rendendo semplice per il programma assemblatore la conversione di questi mnemonici nella loro rappresentazione.

| INDIRIZZO | ISTRUZIONE ESADECIMALE | | | LABEL | CODICE OPERATIVO SIMBOLICO | OPERANDO | COMMENTI |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | | | | |
| | | | | | | | |

Figura 10.5: Formato di programmazione del microprocessore

*Campi dell'assemblatore*

Quando si sta rappresentando un programma per l'assemblatore, si è visto che vengono utilizzati dei campi. Essi sono:

*Il campo della label,* opzionale, che può contenere un indirizzo simbolico per l'istruzione che segue.

*Il campo dell'istruzione,* che comprende il codice operativo e gli operandi. (Può essere distinguibile un campo operando separato).

*Il campo del commento* all'estrema destra, che è opzionale e serve per chiarire il programma.

Una volta che il programma è stato fornito all'assemblatore, quest'ultimo produrrà un suo *listing.* Nella generazione di un listing l'assemblatore fornirà tre campi addizionali, normalmente sulla sinistra della pagina. Un esempio appare di seguito: all'estrema sinistra vi è il numero della riga. Ad ogni riga stampata dal programmatore viene assegnato un numero di riga simbolico.

Il campo successivo a destra è il campo dell'indirizzo effettivo, che mostra in esadecimale il valore del contatore di programma che punterà a quell'istruzione.

Il campo successivo a destra è la rappresentazione esadecimale dell'istruzione.

Questo mostra uno dei possibili impieghi di un assemblatore. Anche se si stanno progettando programmi per un microcomputer su scheda singola che accetta soltanto l'esadecimale si scriverebbe ancora il programma in linguaggio di livello assembly, supponendo di avere accesso ad un sistema equipaggiato di un assemblatore. Si possono quindi inserire i programmi sul sistema utilizzando l'assemblatore. L'assemblatore genererà automaticamente la codifica esadecimale corretta. Quindi si rappresenterà semplicemente in codici esadecimali sul sistema disponibile. Questo mostra, come semplice esempio, il valore delle risorse software addizionali.

*Tabelle*

Quando l'assemblatore traduce il programma simbolico nella sua rappresentazione binaria, esso esegue due compiti essenziali:

1. Esso traduce le istruzioni mnemoniche nella loro codifica binaria.
2. Esso traduce i simboli utilizzati per le costanti e gli indirizzi nella loro rappresentazione binaria.

Per facilitare il collaudo del programma, l'assemblatore indica alla fine del listing l'equivalenza tra i simboli utilizzati ed il loro valore esadecimale. Questo è chiamato: tabella dei simboli.

```
LINEA  #LOC  CODICE          LINEA

0057   0342  A9 00                   LDA   #$ 00
0058   0344  8D 0B A0                STA   ACR1       ; COMMUTA OFF ENTRAMBI
                                                      ; TIMER.
0059   0347  8D 0B AC                STA   ACR2
0060   034A  A2 20                   LDX   # OFFDEL   ; ACCETTA LA COSTANTE
                                                      ; DI RITARDO TONES-OFF.
0061   034C  20 55 03  OFF           JSR   DELAY      ; RITARDO MENTRE TONE
                                                      ; È OFF.
0062   034F  CA                      DEX
0063   0350  D0 FA                   BNE   OFF
0064   0352  4C 02 03                JMP   DIGIT      ; RITORNA AL DIGIT
                                                      ; SUCCESSIVO DEL NUMERO
                                                      ; PHONE.
0065   0355                  ;
0066   0355                  ; QUESTA È UNA SEMPLICE ROUTINE DI RITARDO
                             ; PER IL TONO ON ED OFF.
0067   0355                  ;
0068   0355  A9 FF     DELAY         LDA   * DELCON   ; ACCETTA LA COSTANTE
                                                      ; DI RITARDO.
0069   0357  38        WAIT          SEC              ; RITARDO DI QUESTA
                                                      ; LUNGHEZZA.
0070   0358  E9 01                   SBC   #$ 01
0071   035A  D0 FB                   BNE   WAIT
0072   035C  60                      RTS
0073   035D                  ;
0074   035D                  ; QUESTA È UNA TABELLA DELLE COSTANTI PER
                             ; LE FREQUENZE DI TONO
0075   035D                  ; DI CIASCUNA CIFRA TELEFONICA. LE COSTANTI SONO
                             ; LUNGHE DUE BYTE.
0076   035D                  ; IL PRIMO È IL BYTE DI BASSO ORDINE.
0077   035D                  ;
0078   035D  13        TABLE   *BYTE $13, $2, $76, $ 01     ;   DUE TONI PER "0"
0078   035E  02
0078   035F  76
0078   0360  01
0079   0361  CD               *BYTE $CD, $02, $9E, $ 01    ; DUE TONI PER "1"
0079   0362  02
0079   0363  9E
0079   0364  01
0080   0365  CD                *BYTE $CD, $02, $76, $ 01    ;          '2'
0080   0366  02
0080   0367  76
0080   0368  01
0081   0369  CD                *BYTE $CD, $02, $53, $ 01    ;          '3'
0081   036A  02
0081   036B  53
0081   036C  01
0082   036D  89                *BYTE $89, $02, $9E, $ 01    ;          '4'
0082   036E  02
0082   036E  9E
0082   0370  01
0083   0371  89                *BYTE $89, $02, $76, $ 01    ;          '5'
0083   0372  02
0083   0373  76
0083   0374  01
0084   0375  89                *BYTE $89, $02, $53, $ 01    ;          '6'
0084   0376  02
0084   0377  53
0084   0378  01
0085   0379  4B                *BYTE $4B, $02, $9E, $ 01    ;          '7'
```

Figura 10.6: Output dell'assemblatore: un esempio (continua)

```
0085    037A  02
0085    037B  9E
0085    037C  01
0086    037D  4B                 'BYTE $4B, $02, $76, $ 01    ;        '8'
0086    037E  02


LINEA  #LOC   CODICE LINEA

0086    037E  76
0086    0380  01
0087    0381  4B                 'BYTE $4B, $02, $53, $ 01    ;        '9'
0087    0382  02
0087    0383  53
0087    0384  01
0088    0385                     .END

SYMBOL TABLE

SYMBOL VALUE

ACR1    A00B  ACR2   AC0B  DELAY   0353  DELCON  00FF
DIGIT   0302  NDEND  030A  NUMPTR  0000  OFF     034C
DEFDEL  0020  DN     033C  ONDEL   0040  PHONE   0300
T1CH    A005  T1LH   A007  T1LL    A004  T2CH    AC05
T2LH    AC07  T2LL   AC04  TABLE   035D  WAIT    0357

END OF ASSEMBLY
```

Figura 10.6: Output dell'assemblatore: un esempio

Alcune tabelle dei simboli non solo elencheranno i simboli ed il loro valore ma anche i numeri delle righe dove appaiono i simboli e questa è una caratteristica addizionale.

### *Messaggi di Errore*

Durante il processo assembly, l'assemblatore rileverà errori di sintassi e li elencherà come parte del listing finale. Diagnostici tipici sono: simboli indefiniti, label già definite, codici operativi non consentiti, indirizzi e modi di indirizzamento non consentiti. Naturalmente sono desiderabili diagnostici molto più dettagliati e normalmente vengono forniti. Essi variano da assemblatore ad assemblatore.

### *Il Linguaggio Assembly*

I codici operativi sono già stati definiti. Si definiranno qui i simboli, le costanti e gli operatori che possono essere utilizzati come parte della sintassi dell'assemblatore.

*Simboli*

I simboli sono utilizzati per rappresentare valori numerici, sia dati che indirizzi. Tradizionalmente i simboli comprendono 6 caratteri e devono iniziare con un carattere alfabetico.
Esiste un'ulteriore restrizione: i 56 codici operativi utilizzati dal 6502 oppure i nomi dei registri A, X, Y, S, P possono non essere utilizzati come simboli.

*Assegnazione di un Valore ad un Simbolo*

Le label sono simboli speciali i cui valori non necessitano di essere definiti dal programmatore. Essi corrispondono automaticamente al numero della riga dove esse appaiono. Comunque gli altri simboli utilizzati come costanti od indirizzi di memoria devono essere definiti dal programmatore prima del loro impiego. Il segno uguale è utilizzato per questo scopo od anche come "direttiva" speciale. Esso è un'istruzione all'assemblatore che non sarà tradotto in uno statement eseguibile: essa è chiamata una *direttiva* dell'assemblatore.

Per esempio la costante ALPHA sarà definita:

ALPHA = $A000

Questo assegna il valore "A000" esadecimale alla variabile ALPHA. Le direttive dell'assemblatore saranno esaminate in un paragrafo successivo.

*Costanti o Letterali*

Le costanti possono essere espresse tradizionalmente sia in decimale, oppure in esadecimale, oppure in ottale o in binario.
Per differenziare la base utilizzata per rappresentare un numero viene utilizzato un prefisso. Nel caso di un numero decimale non viene utilizzato il prefisso. Per caricare 18 nell'accumulatore si scriverà semplicemente:

LDA # 18 (dove # denota un letterale)

Un numero esadecimale sarà preceduto dal simbolo $.
Un simbolo ottale sarà preceduto dal simbolo@.
Un simbolo binario sarà preceduto da %.

Per esempio per caricare il valore "11111111" nell'accumulatore si scriverà:

LDA # % 11111111.

I caratteri letterali ASCII possono anche essere utilizzati in un campo letterale. Negli assemblatori più vecchi era tradizionale comprendere il simbolo ASCII tra virgolette. Negli assemblatori più recenti, per avere meno caratteri da stampare, il carattere alfanumerico è indicato da una singola virgoletta che precede il simbolo.

Per esempio per caricare il simbolo "S" nell'accumulatore (in ASCII) si scriverà:

LDA #'S

Per caricare il simbolo delle virgolette stesso la convenzione è:

LDA #'''

**Esercizio 10-1:** *Le due seguenti istruzioni caricheranno lo stesso valore nell'accumulatore: LDA #*'5 ed LDA #$5?

*Operatori*

Per facilitare ulteriormente la scrittura di programmi simbolici, gli assemblatori consentono l'impiego di operatori. al minimo essi dovrebbero consentire l'impiego degli operatori più e meno cosicchè si può specificare per esempio

LDA ADR1, ed:
LDX ADR1 + 1

È importante capire che l'espressione ADR1 +1 sarà calcolato dall'assemblatore per determinare qual'è l'indirizzo di memoria effettivo che deve essere inserito come equivalente binario. Esso sarà calcolato nel *tempo-assembly* e non nel tempo di esecuzione del programma.

Inoltre possono essere disponibili più operatori, come quello di moltiplicazione e divisione, che sono convenienti nell'accesso di tabelle in memoria. Possono essere disponibili anche operatori più specializzati come, per esempio, maggiore o minore di, che troncano un valore di 2 byte rispettivamente nel suo byte di ordine elevato o basso.

Naturalmente un'espressione deve originare un valore positivo. I numeri negativi normalmente possono essere utilizzati e dovrebbero essere espressi in un formato esadecimale.

Infine un simbolo speciale viene tradizionalmente utilizzato per rappresentare il valore attuale dell'indirizzo della riga: *. Questo simbolo dovrebbe essere interpretato come "locazione attuale". (Valore di PC).

**Esercizio 10-2:** *Qual'è la differenza tra le istruzioni seguenti?*
LDA % 10101010
LDA # % 10101010

**Esercizio 10-3:** *Qual'è l'effetto della seguente istruzione?*
BMI* — 2?

*Direttive Dell'Assemblatore*

Le direttive sono ordini speciali dati dal programmatore all'assemblatore, che si risolve nell'immagazzinamento dei valori in simboli o nella memoria ovvero che verranno utilizzate per controllare l'esecuzione dei modi di stampa dell'assemblatore.

Per fornire un esempio specifico si analizzerà qui la nona direttiva dell'assemblatore disponibile sul sistema di sviluppo Rockwell ("System 65"). Questa è:, .BYT, .WOR, .GBY, .PAGE, .SKIP, .OPT, .FILE e END.

*Direttiva di Uguaglianza*

Un segno uguale viene utilizzato per assegnare un valore numerico ad un simbolo. Per esempio:

BASE #$ 1111
* # $ 1234

L'effetto della prima direttiva è di assegnare il valore 1111 esadecimale a BASE.

L'effetto della seconda istruzione è di forzare l'indirizzo della riga al valore esadecimale "1234". In altre parole la successiva istruzione eseguibile incontrata sarà immagazzinata alla locazione di memoria 1234.

**Esercizio 10-4:** *Si scriva una direttiva che causi il trasferimento del programma alla locazione di memoria 0 e successive.*

*Direttive per Inizializzare la Memoria*

Sono disponibili tre direttive per questo scopo: .BYT, .WOR, .GBY.
.BYT assegnerà i caratteri o valori che seguono a byte di memoria consecutivi.

Esempio: RESERV .BYT "SYBEX"

Questo si risolverà nell'immagazzinare gli indirizzi di 2 byte nella memoria, il primo è il byte di basso ordine.

Esempio: .WOR $1234, $2345

.GBY è identico a .WOR eccetto che esso immagazzinerà un valore a 16 bit dove il primo byte è quello di ordine elevato. Esso è normalmente utilizzato per dati a 16 bit piuttosto che per indirizzi a 16 bit.

Le tre direttive successive sono utilizzate per controllare l'ingresso/uscita.

*Direttive d'Ingresso/Uscita*

Esse sono: .PAGE, .SKIP, .OPT.

.PAGE impone all'assemblatore di terminare la pagina, cioè muove alla sommità della pagina successiva. Inoltre può essere specificato un titolo per la pagina.

Esempio: .PAGE "titolo della pagina"

.SKIP è utilizzato per inserire righe bianche nel listing.
Il numero di righe da saltare può essere specificato:
per esempio: .SKIP 3

.OPT specifica quattro scelte: lista, generazione, errori, simbolo. *Lista* genererà una lista. *Generazione* è utilizzato per stampare il codice oggetto di stringhe con la direttiva .BYT. *Errore* specifica se devono essere stampati gli errori diagnostici. *Simbolo* specifica se deve essere elencata la tabella di simbolo.

Le ultime direttive controllano il formato del listing dell'assemblatore.

*Direttive .FILE ed .END.*

Nello sviluppo di un grosso programma, diverse posizioni del programma saranno tipicamente scritte e collaudate separatamente. Ad un certo punto sarà necessario assemblare insieme questi file. L'ultimo statement del primo file comprenderà quindi la direttiva .FILE NAME/1, dove 1 è il numero dell'unità disco e NAME è il nome del file successivo. Il file successivo deve essere collegato, a sua volta, a più file. Alla fine dell'ultimo file ci sarà la direttiva: .END NAME/1 che è un puntatore al primo file.

Infine esiste la possibilità di inserzione di commenti addizionali con il listing ";".

";" può essere utilizzato per far entrare commenti all'interno di una riga piuttosto che far entrare un'istruzione. Questa è una caratteristica importante se i programmi devono essere correttamente documentati.

## MACRO

La caratteristica macro è correttamente non disponibile sugli assemblatori esistenti del 6502. Comunque si definirà qui cos'è una macro e

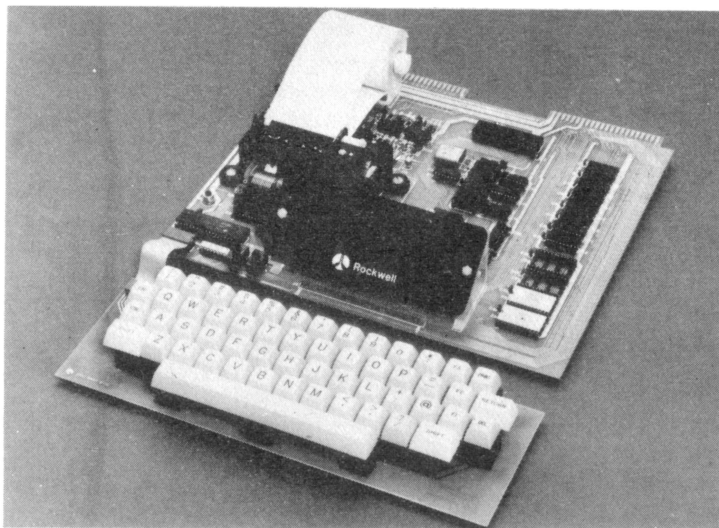



Figura 10.7: L'AIM 65 è una scheda con una Mini-stampante ed una tastiera completa

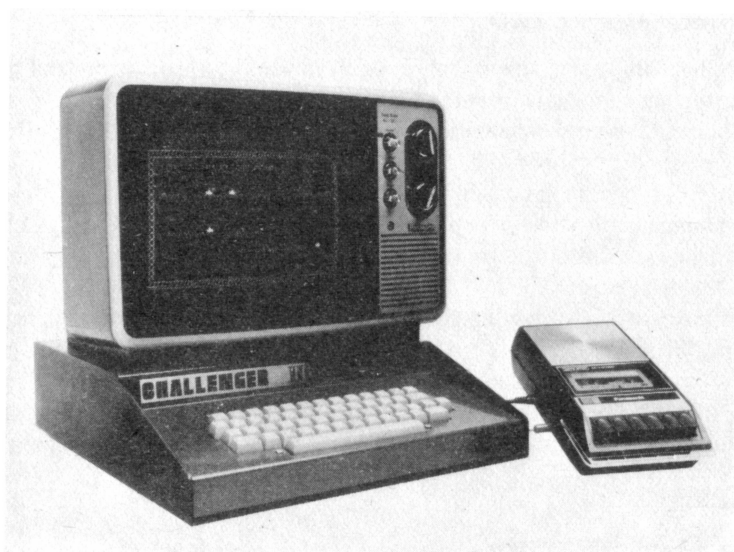



Figura 10.8: Lo Ohio Scientific è un Personal Microcomputer

quali sono i suoi vantaggi. Si spera che la possibilità macro sia presto disponibile sulla maggior parte degli assemblatori del 6502.

Una macro è semplicemente un nome assegnato ad un gruppo di istruzioni. Una macro è essenzialmente una convenienza per il programmatore. Per esempio se un gruppo di cinque istruzioni è utilizzato diverse volte in un programma, si potrebbe definire una macro invece di dover sempre riscrivere queste cinque istruzioni. Come esempio si potrebbe scrivere:

```
SAVREG MACRO        PHA
                    TXA
                    PHA
                    TYA
                    PHA
       ENDM
```

E quindi scrivere il nome: SAVREG invece delle precedenti istruzioni.

Ogni volta che si scrive SAVREG le cinque righe corrispondenti verranno sostituite al posto del nome. Un assemblatore equipaggiato con una caratteristica macro è detto un macro assemblatore. Quando il macro assemblatore incontrerà SAVREG esso eseguirà una vera sostituzione fisica delle righe equivalenti.

### *Macro oppure Subroutine?*

A questo punto una macro può essere vista operare in modo analogo alla subroutine. Questo non è vero. Quando un assemblatore viene impiegato per produrre il codice oggetto, ogni volta che viene incontrato il nome di una macro, essa sarà sostituita dalle istruzioni effettive che compaiono molte volte e che essa sostituisce. Per quanto riguarda il tempo di esecuzione il gruppo di istruzioni apparirà altrettante volte del nome della macro.

In contrapposizione una subroutine è definita soltanto una volta e quindi essa può essere utilizzata ripetutamente: il programma salterà all'indirizzo della subroutine. Una macro è detta una caratteristica di *tempo-assembly.* Una subroutine è una caratteristica di *tempo di esecuzione.* Il loro funzionamento è abbastanza diverso.

### *Parametri della Macro*

Ogni macro può essere equipaggiata di un certo numero di parametri. Per esempio si consideri la macro seguente:

```
SWAP    MACRO   M, N, T
        LDA     M
        STA     T
        LDA     N
        STA     M
        LDA     T
        STA     N
        ENDM
```

Questa macro originerà lo scambio dei contenuti delle locazioni di memoria M ed N. Uno scambio tra due registri, oppure due locazioni di memoria, è un'operazione non disponibile sul 6502. Una macro può essere utilizzata per realizzarla. "T" in questo caso è semplicemente il nome di una locazione di immagazzinamento temporaneo richiesta dal programma. Per esempio si vogliono scambiare i contenuti delle locazioni di memoria ALPHA e BETA. L'istruzione che fa questo appare di seguito: SWAP ALPHA, BETA, TEMP.

In questa istruzione TEMP è il nome di qualche locazione di immagazzinamento temporaneo che si conosce essere disponibile e che può essere utilizzata dalla macro. L'espansione risultante della macro appare di seguito:

```
LDA  ALPHA
STA  TEMP
LDA  BETA
STA  ALPHA
LDA  TEMP
STA  BETA
```

Dovrebbe essere così chiaro il valore di una macro: essa è conveniente per il programmatore per utilizzare le pseudo-istruzioni che sono state definite con macro. In questo modo il set di istruzione apparente del 6502 può essere espanso. Sfortunatamente si deve ricordare che ogni direttiva macro si espanderà in un numero qualsiasi di istruzioni utilizzate. A causa della sua convenienza per lo sviluppo di qualsiasi programma lungo una caratteristica macro è altamente desiderabile per tali applicazioni.

### *Caratteristiche Addizionali della Macro*

Molte altre direttive e caratteristiche sintattiche possono essere aggiunte ad una caratteristica macro semplice: le macro possono essere *annidate,* cioè una chiamata macro può apparire all'interno di una definizione macro. Utilizzando questa caratteristica una macro può modificare sé stessa con una definizione annidata! Una prima chiamata produrrà un'espansione mentre le chiamate successive produrranno un'espansione modificata.

## ASSEMBLY CONDIZIONALE

L'assembly condizionale è un'altra caratteristica dell'assemblatore che fin'ora non è stata fornita sulla maggior parte degli assemblatori del 6502. Una caratteristica di assemblatore condizionale consente al programmatore di utilizzare le istruzioni speciali "IF", seguito da una espressione, quindi (a scelta) "ELSE", e terminata da "ENDIF". Ogni volta che l'espressione seguente l'IF è vera allora verranno assemblate le istruzioni tra l'IF ed LESE oppure IF ed ENDIF (se non c'è "ELSE"). Nel caso in cui sia utilizzato IF seguito da ELSE solo uno dei due blocchi di istruzioni sarà assemblato, dipendentemente dal valore dell'espressione verificata.

Con una caratteristica di assembly condizionale il programma può progettare i programmi per una grande varietà di casi e quindi assemblare condizionalmente i segmenti di codice richiesti da un'applicazione specifica. Per esempio un utente industriale deve progettare programmi che controllino qualsiasi numero di semafori ad un incrocio per una certa varietà di algoritmi di controllo. Esso riceverà quindi le specifiche dall'ingegnere del traffico locale che definiscono il numero di semafori che vi dovrebbero essere e quali algoritmi di controllo. Il programmatore quindi porrà semplicemente i parametri nel suo programma e quindi li assemblerà condizionalmente. L'assembly condizionale si risolverà in un programma " a richiesta" che rivelerà solo quelle routine che sono necessarie per la soluzione del problema.

L'assembly condizionale è perciò di valore specifico per la generazione di programmi industriali in un ambiente dove esistono molte scelte e dove il programmatore desidera assemblare velocemente ed automaticamente porzioni del programma in relazione a parametri esterni.

## SOMMARIO

Questo capitolo ha presentato le tecniche e gli strumenti hardware e software richiesti per sviluppare un programma, insieme ai vari compromessi ed alternative.

Questo a livello hardware va dal microcomputer su scheda singola al sistema di sviluppo completo.

A livello software si va dalla codifica binaria alla programmazione ad alto livello.

Si dovrà quindi operare una selezione in funzione dei traguardi e delle risorse.

CAPITOLO 11

# CONCLUSIONI

Sono stati trattati tutti gli aspetti più importanti della programmazione, dalla definizione e dai concetti di base alla manipolazione interna dei registri del 6502, alla direzione dei dispositivi d'ingresso/uscita, come pure le caratteristiche degli aiuti dello sviluppo software. Qual'è la fase successiva? Si possono presentare due punti di vista, il primo collega lo sviluppo alla tecnologia, il secondo collega lo sviluppo alla propria conoscenza ed abilità. Si indirizzeranno questi due punti.

## SVILUPPO TECNOLOGICO

Il progresso dell'integrazione della tecnologia MOS rende possibile la realizzazione di chip molto più complessi. Il costo di realizzazione della funzione processore stessa è costantemente decrescente. Il risultato è che molti dei chip d'ingresso/uscita o dei chip di controllo di periferica utilizzate in un sistema, ora incorporano un semplice processore. Questo significa che la maggior parte dei chip LSI ora impiegati nel sistema sono divenuti *programmabili.* Si sta sviluppando ora un interessante dilemma concettuale: in modo da semplificare il compito del progetto software come pure di ridurre il numero di componenti i nuovi chip I/O ora comprendono sofisticate caratteristiche programmabili: molti algoritmi programmati sono ora integrati all'interno del chip. Comunque come risultato, lo sviluppo dei programmi è complicato dal fatto che tutti questi chip d'ingresso/uscita sono molto diversi e neessitano di essere studiati in dettaglio dal programmatore! *La programmazione del sistema non è più la programmazione del solo microprocessore, ma anche la programmazione di tutti i vari chip connessi ad esso.* Il tempo di apprendimento per ogni chip può essere significativo.

Naturalmente questo è un dilemma soltanto apparente. Se questi chip non fossero disponibili, la complessità dell'interfacccia da realizzare, come pure i programmi corrispondenti, sarebbe ancora maggiore. La nuova complessità introdotta è che occorre programmare più di un processore ed imparare le varie caratteristiche dei diversi chip di un sistema per rendere effettivo il loro impiego. Comunque si spera che le

Figura 11.1: Il CBM è un sistema di gestione completo con floppy disk e stampante

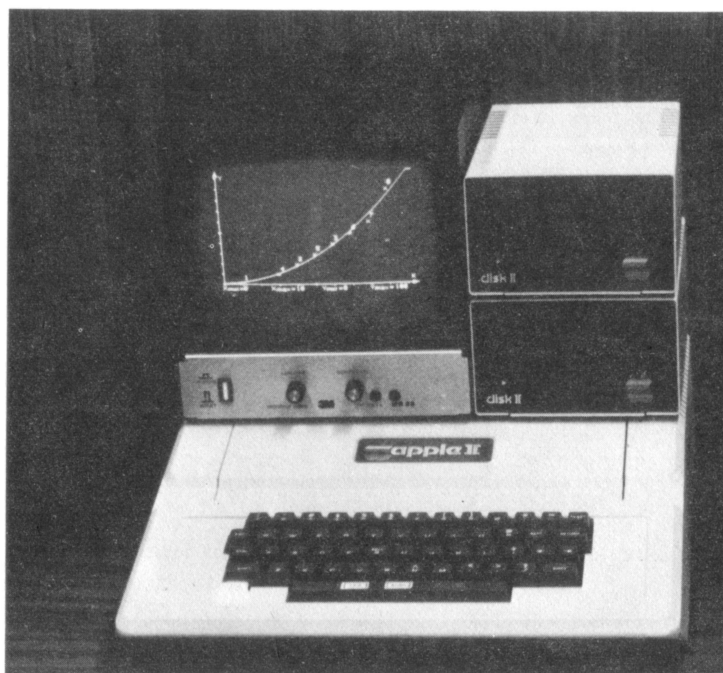



Figura 11.2: L'APPLE II utilizza una TV convenzionale

tecniche ed i concetti presentati in questo libro possano rendere questo compito ragionevolmente semplice.

## LA FASE SUCCESSIVA

Si sono ora imparate le tecniche di base per programmare applicazioni semplici su carta. Questo era il traguardo del libro. La fase successiva è di praticare effettivamente. Non esiste un sostituto a questo. È impossibile imparare completamente la programmazione sulla carta ed è richiesta esperienza. Si dovrebbe quindi ora iniziare la scrittura di programmi propri. Si spera che questa sia una cosa gradita.

Per coloro che desiderano beneficiare della guida di un libro addizionale, il volume complementare a questo in questa serie è: "Applicazioni del 6502" che presenta un insieme di applicazioni effettive che possono essere eseguite su un microcomputer reale.

APPENDICE A

# TABELLA DI CONVERSIONE ESADECIMALE

| HEX | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | 00 | 000 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 0 | 0 |
| 1 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 256 | 4096 |
| 2 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 512 | 8192 |
| 3 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 768 | 12288 |
| 4 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 1024 | 16384 |
| 5 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 1280 | 20480 |
| 6 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 1536 | 24576 |
| 7 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 | 1792 | 28672 |
| 8 | 128 | 129 | 130 | 131 | 132 | 133 | 134 | 135 | 136 | 137 | 138 | 139 | 140 | 141 | 142 | 143 | 2048 | 32768 |
| 9 | 144 | 145 | 146 | 147 | 148 | 149 | 150 | 151 | 152 | 153 | 154 | 155 | 156 | 157 | 158 | 159 | 2304 | 36864 |
| A | 160 | 161 | 162 | 163 | 164 | 165 | 166 | 167 | 168 | 169 | 170 | 171 | 172 | 173 | 174 | 175 | 2560 | 40960 |
| B | 176 | 177 | 178 | 179 | 180 | 181 | 182 | 183 | 184 | 185 | 186 | 187 | 188 | 189 | 190 | 191 | 2816 | 45056 |
| C | 192 | 193 | 194 | 195 | 196 | 197 | 198 | 199 | 200 | 201 | 202 | 203 | 204 | 205 | 206 | 207 | 3072 | 49152 |
| D | 208 | 209 | 210 | 211 | 212 | 213 | 214 | 215 | 216 | 217 | 218 | 219 | 220 | 221 | 222 | 223 | 3328 | 53248 |
| E | 224 | 225 | 226 | 227 | 228 | 229 | 230 | 231 | 232 | 233 | 234 | 235 | 236 | 237 | 238 | 239 | 3584 | 57344 |
| F | 240 | 241 | 242 | 243 | 244 | 245 | 246 | 247 | 248 | 249 | 250 | 251 | 252 | 253 | 254 | 255 | 3840 | 61440 |

| 5 | | 4 | | 3 | | 2 | | 1 | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| HEX | DEC | HEX | DEC | HEX | DEC | HEX | DEC | HEX | DEC | HEX | DEC |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1,048,576 | 1 | 65,536 | 1 | 4,096 | 1 | 256 | 1 | 16 | 1 | 1 |
| 2 | 2,097,152 | 2 | 131,072 | 2 | 8,192 | 2 | 512 | 2 | 32 | 2 | 2 |
| 3 | 3,145,728 | 3 | 196,608 | 3 | 12,288 | 3 | 768 | 3 | 48 | 3 | 3 |
| 4 | 4,194,304 | 4 | 262,144 | 4 | 16,384 | 4 | 1,024 | 4 | 64 | 4 | 4 |
| 5 | 5,242,880 | 5 | 327,680 | 5 | 20,480 | 5 | 1,280 | 5 | 80 | 5 | 5 |
| 6 | 6,291,456 | 6 | 393.216 | 6 | 24,576 | 6 | 1,536 | 6 | 96 | 6 | 6 |
| 7 | 7,340,032 | 7 | 458,752 | 7 | 28,672 | 7 | 1,792 | 7 | 112 | 7 | 7 |
| 8 | 8,388,608 | 8 | 524,288 | 8 | 32,768 | 8 | 2,048 | 8 | 128 | 8 | 8 |
| 9 | 9,437,184 | 9 | 589,824 | 9 | 36,864 | 9 | 2,304 | 9 | 144 | 9 | 9 |
| A | 10,485,760 | A | 655,360 | A | 40,960 | A | 2,560 | A | 160 | A | 10 |
| B | 11,534,336 | B | 720,896 | B | 45,056 | B | 2,816 | B | 176 | B | 11 |
| C | 12,582,912 | C | 786,432 | C | 49,152 | C | 3,072 | C | 192 | C | 12 |
| D | 13,631,488 | D | 851,968 | D | 53,248 | D | 3,328 | D | 208 | D | 13 |
| E | 14,680,064 | E | 917,504 | E | 57,344 | E | 3,584 | E | 224 | E | 14 |
| F | 15,728,640 | F | 983,040 | F | 61,440 | F | 3,840 | F | 240 | F | 15 |

APPENDICE B

# ISTRUZIONI IN ORDINE ALFABETICO DEL 6502

| | |
|---|---|
| **ADC** | Somma con riporto |
| **AND** | AND Logico |
| **ASL** | Spostamento Aritmetico a Sinistra |
| **BCC** | Opera diramazione se carry è zero |
| **BCS** | Opera diramazione se carry è uno |
| **BEQ** | Opera diramazione se risultato = 0 |
| **BIT** | Verifica di bit |
| **BMI** | Opera diramazione se negativo |
| **BNE** | Opera diramazione se diverso da 0 |
| **BPL** | Opera diramazione se positivo |
| **BRK** | Break |
| **BVC** | Opera diramazione se overflow è 0 |
| **BVS** | Opera diramazione se overflow è 1 |
| **CLC** | Azzera carry |
| **CLD** | Azzera il flag decimale |
| **CLI** | Azzera la disabilitazione interrupt |
| **CLV** | Azzera overflow |
| **CMP** | Confronta con l'accumulatore |
| **CPX** | Confronta con X |
| **CPY** | Confronta con Y |
| **DEC** | Decrementa la memoria |
| **DEX** | Decrementa X |
| **DEY** | Decrementa Y |
| **EOR** | OR Esclusivo |
| **INC** | Incrementa X |
| **INY** | Incrementa Y |
| **JMP** | Salta |
| **JSR** | Salta alla subroutine |
| **LDA** | Carica l'accumulatore |
| **LDX** | Carica X |
| **LDY** | Carica Y |
| **LSR** | Spostamento logico a destra |
| **NOT** | Non opera |
| **ORA** | OR Logico |
| **PHA** | Introduce A |
| **PHP** | Introduce lo stato P |
| **PLA** | Estrae A |
| **PLP** | Estrae lo stato P |
| **ROL** | Rotazione a sinistra |
| **ROR** | Rotaziona a destra |
| **RTI** | Ritorno da Interrupt |
| **RTS** | Ritorno da subroutine |
| **SBC** | Sottrae con riporto |
| **SEC** | Pone carry ad 1 |
| **SED** | Pone decimale ad 1 |
| **SEI** | Pone disabilitazione interrupt ad 1 |
| **STA** | Immagazzina l'accumulatore |
| **STX** | Immagazzina X |
| **STY** | Immagazzina Y |
| **TAX** | Trasferisce A in X |
| **TAY** | Trasferisce A in Y |
| **TSX** | Trasferisce SP in X |
| **TXA** | Trasferisce X in A |
| **TXS** | Trasferisce X in SP |
| **TYA** | Trasferisce Y in A |

APPENDICE C

# LISTING BINARIO DELLE ISTRUZIONI DEL 6502

| | | | |
|---|---|---|---|
| ADC | 011bbb01 | LDX | 101bbb10 |
| AND | 001bbb01 | LDY | 101bbb00 |
| ASL | 000bbb10 | LSR | 01bbbb10 |
| BCC | 10110000 | NOP | 01bbb110 |
| BEQ | 11110000 | ORA | 000bbb01 |
| BIT | 0010b100 | PHP | 01001000 |
| BMI | 00110000 | PHP | 00001000 |
| BNE | 11010000 | PLA | 01101000 |
| BPL | 00010000 | PLP | 00101000 |
| BRK | 01010000 | ROL | 001bbb10 |
| CLC | 00011000 | ROR | 011bbb10 |
| CLD | 11011000 | RTI | 01000000 |
| CLI | 01011000 | RTS | 01100000 |
| CMP | 110bbb01 | SBC | 111bbb01 |
| CPX | 1110bb00 | SEC | 00111000 |
| CPY | 1100bb00 | SED | 11111000 |
| DEC | 110bb110 | SEI | 01111000 |
| DEX | 11001010 | STA | 100bbb01 |
| DEY | 10001000 | STX | 100bb110 |
| EDR | 101bbb01 | STY | 100bb100 |
| INC | 111bb110 | TAX | 10101010 |
| INX | 11101000 | TAY | 10101000 |
| INY | 11001000 | TSX | 10111010 |
| JMP | 01b01100 | TXA | 10001010 |
| JSR | 00100000 | TXS | 10011010 |
| LDA | 101bbb01 | TYA | 10011000 |

Per la definizione del campo “bb” si faccia riferimento al Capitolo 4

## APPENDICE D

# SET DI ISTRUZIONI DEL 6502: ESADECIMALE E TIMING

| | | IMPLICATO | | | ACC. | | | ASSOLUTO | | | PAGINA 0 | | | IMMEDIATO | | | ABS X | | | ABS Y | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MNEMONICO | | OP | n | # | OP | n | # | OP | n | # | OP | n | # | OP | n | # | OP | n | # | OP | n | # |
| ADC | (1) | | | | | | | 6D | 4 | 3 | 65 | 3 | 2 | 69 | 2 | 2 | 7D | 4 | 3 | 79 | 4 | 3 |
| AND | (1) | | | | | | | 2D | 4 | 3 | 25 | 3 | 2 | 29 | 2 | 2 | 3D | 4 | 3 | 39 | 4 | 3 |
| ASL | | | | | OA | 2 | 1 | OE | 6 | 3 | 06 | 5 | 2 | | | | 1E | 7 | 3 | | | |
| BCC | (2) | | | | | | | | | | | | | | | | | | | | | |
| BCS | (2) | | | | | | | | | | | | | | | | | | | | | |
| BEQ | (2) | | | | | | | | | | | | | | | | | | | | | |
| BIT | | | | | | | | 2C | 4 | 3 | 24 | 3 | 2 | | | | | | | | | |
| BMI | (2) | | | | | | | | | | | | | | | | | | | | | |
| BNE | (2) | | | | | | | | | | | | | | | | | | | | | |
| BPL | (2) | | | | | | | | | | | | | | | | | | | | | |
| BRK | | 00 | 7 | 1 | | | | | | | | | | | | | | | | | | |
| BVC | (2) | | | | | | | | | | | | | | | | | | | | | |
| BVS | (2) | | | | | | | | | | | | | | | | | | | | | |
| CLC | | 18 | 2 | 1 | | | | | | | | | | | | | | | | | | |
| CLD | | D8 | 2 | 1 | | | | | | | | | | | | | | | | | | |
| CLI | | 58 | 2 | 1 | | | | | | | | | | | | | | | | | | |
| CLV | | B8 | 2 | 1 | | | | | | | | | | | | | | | | | | |
| CMP | | | | | | | | CD | 4 | 3 | C5 | 3 | 2 | C9 | 2 | 2 | DD | 4 | 3 | D9 | 4 | 3 |
| CPX | | | | | | | | EC | 4 | 3 | E4 | 3 | 2 | EO | 2 | 2 | | | | | | |
| CPY | | | | | | | | CC | 4 | 3 | C4 | 3 | 2 | CO | 2 | 2 | | | | | | |
| DEC | | | | | | | | CE | 6 | 3 | C6 | 5 | 2 | | | | DE | 7 | 3 | | | |
| DEX | | CA | 2 | 1 | | | | | | | | | | | | | | | | | | |
| DEY | | 88 | 2 | 1 | | | | | | | | | | | | | | | | | | |
| EOR | (1) | | | | | | | 4D | 4 | 3 | 45 | 3 | 2 | 49 | 2 | 2 | 5D | 4 | 3 | 59 | 4 | 3 |
| INC | | | | | | | | EE | 6 | 3 | E6 | 5 | 2 | | | | FE | 7 | 3 | | | |
| INX | | E8 | 2 | 1 | | | | | | | | | | | | | | | | | | |
| INY | | C8 | 2 | 1 | | | | | | | | | | | | | | | | | | |
| JMP | | | | | | | | 4C | 3 | 3 | | | | | | | | | | | | |
| JSR | | | | | | | | 20 | 6 | 3 | | | | | | | | | | | | |
| LDA | (1) | | | | | | | AD | 4 | 3 | A5 | 3 | 2 | A9 | 2 | 2 | BD | 4 | 3 | B9 | 4 | 3 |
| LDX | (1) | | | | | | | AE | 4 | 3 | A6 | 3 | 2 | A2 | 2 | 2 | | | | BE | 4 | 3 |
| LDY | (1) | | | | | | | AC | 4 | 3 | A4 | 3 | 2 | A0 | 2 | 2 | BC | 4 | 3 | | | |
| LSR | | | | | 4A | 2 | 1 | 4E | 6 | 3 | 46 | 5 | 2 | | | | 5E | 7 | 3 | | | |
| NOP | | EA | 2 | 1 | | | | | | | | | | | | | | | | | | |
| ORA | | | | | | | | 0D | 4 | 3 | 05 | 3 | 2 | 09 | 2 | 2 | 1D | 4 | 3 | 19 | 4 | 3 |
| PHA | | 48 | 3 | 1 | | | | | | | | | | | | | | | | | | |
| PHP | | 08 | 3 | 1 | | | | | | | | | | | | | | | | | | |
| PLA | | 68 | 4 | 1 | | | | | | | | | | | | | | | | | | |
| PLP | | 28 | 4 | 1 | | | | | | | | | | | | | | | | | | |
| ROL | | | | | 2A | 2 | 1 | 2E | 6 | 3 | 26 | 5 | 2 | | | | 3E | 7 | 3 | | | |
| ROR | | | | | 6A | 2 | 1 | 6E | 6 | 3 | 66 | 5 | 2 | | | | 7E | 7 | 3 | | | |
| RTI | | 40 | 6 | 1 | | | | | | | | | | | | | | | | | | |
| RTS | | 60 | 6 | 1 | | | | | | | | | | | | | | | | | | |
| SBC | (1) | | | | | | | ED | 4 | 3 | E5 | 3 | 2 | E9 | 2 | 2 | FD | 4 | 3 | F9 | 4 | 3 |
| SEC | | 38 | 2 | 1 | | | | | | | | | | | | | | | | | | |
| SED | | F8 | 2 | 1 | | | | | | | | | | | | | | | | | | |
| SEI | | 78 | 2 | 1 | | | | | | | | | | | | | | | | | | |
| STA | | | | | | | | 8D | 4 | 3 | 85 | 2 | | | | | 9D | 5 | 3 | 99 | 5 | 3 |
| STX | | | | | | | | 8E | 4 | 3 | 86 | 2 | | | | | | | | | | |
| STY | | | | | | | | 8C | 4 | 3 | 84 | 2 | | | | | | | | | | |
| TAX | | AA | 2 | 1 | | | | | | | | | | | | | | | | | | |
| TAY | | A8 | 2 | 1 | | | | | | | | | | | | | | | | | | |
| TSX | | BA | 2 | 1 | | | | | | | | | | | | | | | | | | |
| TXA | | 8A | 2 | 1 | | | | | | | | | | | | | | | | | | |
| TXS | | 9A | 2 | 1 | | | | | | | | | | | | | | | | | | |
| TYA | | 98 | 2 | 1 | | | | | | | | | | | | | | | | | | |

(1) Somma 1 ad n se si sta attraversando il confine di una pagina

| (IND X) | | | (IND)Y | | | PAGINA Z,X | | | RELATIVO | | | INDIRETTO | | | PAGINA Z,Y | | | CODICE DI STATO DEL PROCESSORE | | | | | | | | MNEMONIC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OP | n | # | OP | n | # | OP | n | # | OP | n | # | OP | n | # | OP | n | # | N | V | | B | D | I | Z | C | MNEMONIC |
| 61 | 6 | 2 | 71 | 5 | 2 | 75 | 4 | 2 | | | | | | | | | | ● | ● | | | | | ● | ● | ADC |
| 21 | 6 | 2 | 31 | 5 | 2 | 35 | 4 | 2 | | | | | | | | | | ● | | | | | | ● | | AND |
| | | | | | | 16 | 6 | 2 | | | | | | | | | | ● | | | | | | ● | ● | ASL |
| | | | | | | | | | 90 | 2 | 2 | | | | | | | | | | | | | | | BCC |
| | | | | | | | | | BO | 2 | 2 | | | | | | | | | | | | | | | BCS |
| | | | | | | | | | FO | 2 | 2 | | | | | | | | | | | | | | | BEQ |
| | | | | | | | | | | | | | | | | | | $M_7$ | $M_6$ | | | | | ● | | BIT |
| | | | | | | | | | 30 | 2 | 2 | | | | | | | | | | | | | | | BMI |
| | | | | | | | | | DO | 2 | 2 | | | | | | | | | | | | | | | BNE |
| | | | | | | | | | 10 | 2 | 2 | | | | | | | | | | | | | | | BPL |
| | | | | | | | | | | | | | | | | | | | | | 1 | | 1 | | | BRK |
| | | | | | | | | | 50 | 2 | 2 | | | | | | | | | | | | | | | BVC |
| | | | | | | | | | 70 | 2 | 2 | | | | | | | | | | | | | | | BVS |
| | | | | | | | | | | | | | | | | | | | | | | | | | 0 | CLC |
| | | | | | | | | | | | | | | | | | | | | | | 0 | | | | CLD |
| | | | | | | | | | | | | | | | | | | | | | | | 0 | | | CLI |
| | | | | | | | | | | | | | | | | | | | 0 | | | | | | | CLV |
| C1 | 6 | 2 | D1 | 5 | 2 | D5 | 4 | 2 | | | | | | | | | | ● | | | | | | ● | ● | CMP |
| | | | | | | | | | | | | | | | | | | ● | | | | | | ● | ● | CPX |
| | | | | | | | | | | | | | | | | | | ● | | | | | | ● | ● | CPY |
| | | | | | | D6 | 6 | 2 | | | | | | | | | | ● | | | | | | ● | | DEC |
| | | | | | | | | | | | | | | | | | | ● | | | | | | ● | | DEX |
| | | | | | | | | | | | | | | | | | | ● | | | | | | ● | | DEY |
| 41 | 6 | 2 | 51 | 5 | 2 | 55 | 4 | 2 | | | | | | | | | | ● | | | | | | ● | | EOR |
| | | | | | | F6 | 6 | 2 | | | | | | | | | | ● | | | | | | ● | | INC |
| | | | | | | | | | | | | | | | | | | ● | | | | | | ● | | INX |
| | | | | | | | | | | | | | | | | | | ● | | | | | | ● | | INY |
| | | | | | | | | | | | | 6C | 5 | 3 | | | | | | | | | | | | JMP |
| | | | | | | | | | | | | | | | | | | | | | | | | | | JSR |
| A1 | 6 | 2 | B1 | 5 | 2 | B5 | 4 | 2 | | | | | | | | | | ● | | | | | | ● | | LDA |
| | | | | | | | | | | | | | | | B6 | 4 | 2 | ● | | | | | | ● | | LDX |
| | | | | | | B4 | 4 | 2 | | | | | | | | | | ● | | | | | | ● | | LDY |
| | | | | | | 56 | 6 | 2 | | | | | | | | | | 0 | | | | | | ● | ● | LSR |
| | | | | | | | | | | | | | | | | | | | | | | | | | | NOP |
| 01 | 6 | 2 | 11 | 5 | 2 | 15 | 4 | 2 | | | | | | | | | | ● | | | | | | ● | | ORA |
| | | | | | | | | | | | | | | | | | | | | | | | | | | PHA |
| | | | | | | | | | | | | | | | | | | | | | | | | | | PHP |
| | | | | | | | | | | | | | | | | | | ● | | | | | | ● | | PLA |
| | | | | | | | | | | | | | | | | | | ● | ● | ● | ● | ● | ● | ● | ● | PLP |
| | | | | | | 36 | 6 | 2 | | | | | | | | | | ● | | | | | | ● | ● | ROL |
| | | | | | | 76 | 6 | 2 | | | | | | | | | | ● | | | | | | ● | ● | ROR |
| | | | | | | | | | | | | | | | | | | ● | ● | ● | ● | ● | ● | ● | ● | RTI |
| | | | | | | | | | | | | | | | | | | | | | | | | | | RTS |
| E1 | 6 | 2 | F1 | 5 | 2 | F5 | 4 | 2 | | | | | | | | | | ● | ● | | | | | ● | ● | SBC |
| | | | | | | | | | | | | | | | | | | | | | | | | | 1 | SEC |
| | | | | | | | | | | | | | | | | | | | | | | 1 | | | | SED |
| | | | | | | | | | | | | | | | | | | | | | | | 1 | | | SEI |
| 81 | 6 | 2 | 91 | 6 | 2 | 95 | 4 | 2 | | | | | | | | | | | | | | | | | | STA |
| | | | | | | | | | | | | | | | 96 | 4 | 2 | | | | | | | | | STX |
| | | | | | | 94 | 4 | 2 | | | | | | | | | | | | | | | | | | STY |
| | | | | | | | | | | | | | | | | | | ● | | | | | | ● | | TAX |
| | | | | | | | | | | | | | | | | | | ● | | | | | | ● | | TAY |
| | | | | | | | | | | | | | | | | | | ● | | | | | | ● | | TSX |
| | | | | | | | | | | | | | | | | | | ● | | | | | | ● | | TXA |
| | | | | | | | | | | | | | | | | | | | | | | | | | | TXS |
| | | | | | | | | | | | | | | | | | | ● | | | | | | ● | | TYA |

(2) Somma 2 ad n se si ha diramazione all'interno della pagina
Somma 3 ad n se si ha diramazione ad un'altra pagina

## APPENDICE E

# TABELLA DI CONVERSIONE ASCII

| HEX | MSD | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|
| LSD | BITS | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| 0 | 0000 | NUL | DLE | SPACE | 0 | @ | P | – | p |
| 1 | 0001 | SOH | DC1 | ! | 1 | A | Q | a | q |
| 2 | 0010 | STX | DC2 | " | 2 | B | R | b | r |
| 3 | 0011 | ETX | DC3 | # | 3 | C | S | c | s |
| 4 | 0100 | EOT | DC4 | $ | 4 | D | T | d | t |
| 5 | 0101 | ENQ | NAK | % | 5 | E | U | e | u |
| 6 | 0110 | ACK | SYN | & | 6 | F | V | f | v |
| 7 | 0111 | BEL | ETB | ' | 7 | G | W | g | w |
| 8 | 1000 | BS | CAN | ( | 8 | H | X | h | x |
| 9 | 1001 | HT | EM | ) | 9 | I | Y | i | y |
| A | 1010 | LF | SUB | * | : | J | Z | j | z |
| B | 1011 | VT | ESC | + | ; | K | [ | k | { |
| C | 1100 | FF | FS | , | < | L | \ | l | - - |
| D | 1101 | CR | GS | – | = | M | ] | m | } |
| E | 1110 | SO | RS | . | > | N | ∧ | n | ~ |
| F | 1111 | SI | US | / | ? | O | ← | o | DEL |

## I SIMBOLI ASCII

**NUL** — **Nullo**
**SOH** — **Inizio della testata**
**STX** — **Inizio del testo**
**ETX** — **Fine del Testo**
**EOT** — **Fine della trasmissione**
**ENQ** — **Domanda**
**ACK** — **Riconoscimento**
**BEL** — **Campana**
**BS** — **Spazio posteriore**
**HT** — **Tabulazione orizzontale**
**LF** — **Incremento di riga**
**VT** — **Tabulazione verticale**
**FF** — **Alimentazione del modulo**
**CR** — **Ritorno carrello**
**SO** — **Sposta fuori**
**SI** — **Sposta dentro**
**DLE** — **Perdita Collegamento Dati**
**DC** — **Controllo dispositivo**
**NAK** — **Riconoscimento negativo**
**SYN** — **Sincronismo**
**ETB** — **Fine del blocco di trasmissione**
**CAN** — **Cancella**
**EM** — **Fine del mezzo**
**SUB** — **Sostituto**
**ESC** — **Perdita**
**FS** — **Separatore di file**
**GS** — **Separatore di gruppo**
**RS** — **Separatore di record**
**US** — **Separatore di unità**
**SP** — **Spazio (Blank)**
**DEL** — **Cancella sostituendo**

# APPENDICE F

# TABELLA DELLE DIRAMAZIONI RELATIVE

DIRAMAZIONE RELATIVA DIRETTA

| LSD / MSD | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 1 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 2 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 3 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| 4 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
| 5 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 |
| 6 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
| 7 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 |

TABELLA DI DIRAMAZIONE RELATIVA INVERSA

| LSD / MSD | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 128 | 127 | 126 | 125 | 124 | 123 | 122 | 121 | 120 | 119 | 118 | 117 | 116 | 115 | 114 | 113 |
| 9 | 112 | 111 | 110 | 109 | 108 | 107 | 106 | 105 | 104 | 103 | 102 | 101 | 100 | 99 | 98 | 97 |
| A | 96 | 95 | 94 | 93 | 92 | 91 | 90 | 89 | 88 | 87 | 86 | 85 | 84 | 83 | 82 | 81 |
| B | 80 | 79 | 78 | 77 | 76 | 75 | 74 | 73 | 72 | 71 | 70 | 69 | 68 | 67 | 66 | 65 |
| C | 64 | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 |
| D | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 |
| E | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 |
| F | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

## APPENDICE G

# LISTING DEL CODICE OPERATIVO ESADECIMALE

| LSD / MSD | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | BRK | ORA·I, X | | | | ORA·Φ·P | ASL·Φ·P | |
| 1 | BPL | ORA·I, Y | | | | ORA·Φ·P, X | ASL·Φ·P, X | |
| 2 | JSR | AND·I, X | | | BIT Φ P | AND Φ P | ROL Φ·F | |
| 3 | BMI | AND·I, Y | | | | AND Φ P, X | ROL Φ·P, X | |
| 4 | RTI | EOR·I, X | | | | EOR Φ P | LSR·Φ P | |
| 5 | BVC | EOR·I, Y | | | | EOR Φ P, X | LSR Φ P, X | |
| 6 | RTS | ADC·I, X | | | | ADC Φ·P | ROR Φ P | |
| 7 | BVS | ADC I, Y | | | | ADC Φ P, X | | |
| 8 | | STA·I, X | | | STY·Φ·P | STA·Φ·P | STX·Φ·P | |
| 9 | BCC | STA·I, Y | | | STY Φ·P, X | STA Φ P, X | STX·Φ·P, Y | |
| A | LDY·IMM | LDA·I, X | LDX IMM | | LDY·Φ·P | LDA·Φ·P | LDX·Φ·P | |
| B | BCS | LDA I, Y | | | LDY·Φ·P, X | LDA Φ P, X | LDX·Φ·P, Y | |
| C | CPY·IMM | CMP I, X | | | CPY Φ P | CMP Φ P | DEC·Φ·P | |
| D | BNE | CMP I, Y | | | | CMP Φ P, X | DEC·Φ·P, X | |
| E | CPX·IMM | SBC·I, X | | | CPX Φ P | SBC Φ·P | INC·Φ·P | |
| F | BEQ | SBC I Y | | | | SBC Φ P X | INC Φ·P, X | |

| 8 | 9 | A | B | C | D | E | F | LSD / MSD |
|---|---|---|---|---|---|---|---|---|
| PHP | ORA·IMM | ASL A | | | ORA | ASL | | 0 |
| CLC | ORA, Y | | | | ORA, X | ASL, X | | 1 |
| PLP | AND·IMM | ROL·A | | BIT | AND | ROL | | 2 |
| SEC | AND Y | | | | AND, X | ROL, X | | 3 |
| PHA | EOR·IMM | LSR·A | | JMP | EOR | LSR | | 4 |
| CLI | EOR, Y | | | | EOR, X | LSR, X | | 5 |
| PLA | ADC·IMM | ROR A | | JMP·I | ADC | ROR | | 6 |
| SEI | ADC, Y | | | | ADC, X | | | 7 |
| DEY | | TXA | | STY | STA | STX | | 8 |
| TYA | STA, Y | TXS | | | STA, X | | | 9 |
| TAY | LDA·IMM | TAX | | LDY | LDA | LDX | | A |
| CLV | LDA, Y | TSX | | LDY, X | LDA, X | LDX, Y | | B |
| INY | CMP·IMM | DEX | | CPY | CMP | DEC | | C |
| CLD | CMP, Y | | | | CMP, X | DEC, X | | D |
| INX | SBC IMM | NOP | | CPX | SBC | INC | | E |
| SED | SBC, Y | | | | SBC, X | INC, X | | F |

I = indiretto

0 - P = pagina zero

APPENDICE H

# CONVERSIONE DA DECIMALE A BCD

| DECIMALE | BCD | DEC | BCD | DEC | BCD |
|---|---|---|---|---|---|
| 0 | 0000 | 10 | 00010000 | 90 | 10010000 |
| 1 | 0001 | 11 | 00010001 | 91 | 10010001 |
| 2 | 0010 | 12 | 00010010 | 92 | 10010010 |
| 3 | 0011 | 13 | 00010011 | 93 | 10010011 |
| 4 | 0100 | 14 | 00010100 | 94 | 10010100 |
| 5 | 0101 | 15 | 00010101 | 95 | 10010101 |
| 6 | 0110 | 16 | 00010110 | 96 | 10010110 |
| 7 | 0111 | 17 | 00010111 | 97 | 10010111 |
| 8 | 1000 | 18 | 00011000 | 98 | 10011000 |
| 9 | 1001 | 19 | 00011001 | 99 | 10011001 |

L. 25.000

Cod. 503 B

**L'AUTORE**

Ha insegnato microprocessori e programmazione a più di 5000 persone in tutto il mondo. Laureatosi in Fisica, in Scienza del Calcolatore, all'Università di Berkeley, ha sviluppato una realizzazione APL microprogrammata ed ha lavorato nella Silicon Valley sui sistemi industriali a microprocessore all'inizio della loro comparsa. Questo libro, come gli altri di questa serie, è basato sulla sua esperienza tecnica ed educativa.

**Rodnay Zaks**

50